FRONT-END-KEHITYKSEN PARHAAT KÄYTÄNNÖT

Esa Taskinen

Opinnäytetyö
Lokakuu 2013
Tietojenkäsittely
Digimedia

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittely
Digimedia

TASKINEN ESA:
Front-end-kehityksen parhaat käytännöt

Opinnäytetyö 89 sivua, joista liitteitä 22 sivua
Lokakuu 2013

Opinnäytetyön tavoitteena oli yhtenäistää toimeksiantajan, CGI Suomi Oy:n tulevissa projekteissa verkkosivustojen ja -sovellusten front-end-koodia, parantaa sen ylläpidettävyyttä, nopeuttaa jatkokehitystä ja helpottaa uusien kehittäjien liittymistä tiimiin. Tiimi toteuttaa pääasiassa kaupan alan keskisuurten tai suurten yritysten verkkopalveluita. Tarkoituksena oli laatia ohjelmistokehittäjille yhteisiä käytäntöjä ja ohjeita oppaaseen. Opas käsittelee front-end-kehityksen eri osa-alueita: HTML-merkkausta, CSS-tyylejä, JavaScript-koodausta. Lisäksi määriteltiin sivustojen ja sovellusten selaintukeen suorituskyvyn optimointiin ja esteettömyyteen liittyviä käytäntöjä.

Opinnäytetyö toimii kuvauksena oppaaseen kirjatuista käytännöistä, ohjeista ja tekniikoista. Tarkemmat selitykset rajattiin pois oppaasta, jotta se pysyisi tiiviinä ja kohderyhmälle soveltuvana. Opas on onnistunut tiimin kehittäjien komment-tien perusteella hyvin. He ovat kiinnostuneita aiheista ja toivovat lisää sisältöä jatkossa. Se jää kuitenkin vielä nähtäväksi, kuinka paljon käytäntöjä ja ohjeita tullaan hyödyntämään.

Opas toimii tällä hetkellä hyvänä lähtökohtana sivustojen ja sovellusten front-end-kehitykselle, mutta tarvetta on selvästi vielä yksityiskohtaisemmille, projek-tikohtaisille ohjeille. Siksi oppaan päivittämisen kannalta on tärkeää siirtää se pois dokumentista omaksi sivustokseen. Toinen kehitysmahdollisuus on räätä-löidä opas muiden tiimien tarpeisiin sopivaksi.

Asiasanat: WWW-kehitys, front-end-kehitys, WWW-suunnittelu, HTML, CSS, JavaScript.

ABSTRACT

Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Option of Digital Media

TASKINEN ESA:
Best Practices of Front-End Development

Bachelor's thesis 89 pages, appendices 22 pages
October 2013

The goal of this thesis was to help the applicant, CGI Suomi Oy, in creating better websites and web applications through best practices of front-end development. The main concerns were maintainability, faster development time and easier orientation for new developers joining the team, whose clients are most of the time medium to large size companies in the retail business. To achieve the goal, a guide was written containing practices, guidelines and descriptions of different techniques regarding aspects of front-end development. Those include HTML, CSS and JavaScript coding, cross-browser compatibility, accessibility and performance optimization.

The thesis contains descriptions of the guide's contents and explanations why certain practices were included in it. Most of the detailed information was excluded from the guide to keep it relatively short and simple. What front-end development is and how the ever-increasing mobile usage of the web has affected the guide and the guides' writing process are introduced in the first chapters. Rest of the content is about front-end coding and its techniques.

The guide was well received within the team. The developers were interested in the content and requested future expansions. The next step will be to recreate the guide as an HTML-based documentation. In its current form, it provides a good base for front-end development, but is too general and lacks much-needed documentation about different projects' practices and guidelines.

Key words: web development, front-end development, web design, HTML, CSS, JavaScript.

SISÄLLYS

1 JOHDANTO

Tutustuin WWW-kehitykseen ensimmäistä kertaa datanomiksi opiskellessani. Olen siitä lähtien ollut kiinnostunut alasta, nimenomaan verkkosivustojen ja sovellusten selaimessa toimivasta osasta, niiden käyttöliittymistä eli front-end-puolesta. Olen seurannut, kuinka front-end-puolen tekniikat ja työkalut ovat kehittyneet jatkuvasti. Selaimissa voidaan ajaa kokonaisia sovelluksia, joiden toiminnoista suurin osa on kehitetty JavaScript-kielellä (esim. Googlen Docs-palvelut). Moninaisia työkaluja, kirjastoja ja sovelluskehyksiä hyödyntämällä voidaan lyhentää sivustojen ja sovellusten kehitysaikaa merkittävästi. Alan kehityksestä johtuen pidän jo aiempaa tärkeämpänä pysyä ajan tasalla muutoksissa sekä perehtyä alan parhaisiin käytäntöihin.

Tämä opinnäytetyö on jaettu kahteen osaan: raporttiin ja liitteenä olevaan oppaaseen (Liite 1). Raportti on rajattu käsittelemään toimeksiantajalle laaditun oppaan aiheita. Useimmat aiheista ovat niin laajoja, että vaatisivat oman raporttinsa aiheen kattavaan käsittelyyn. Siksi opinnäytetyö onkin tarkoitettu yleiskuvaukseksi front-end-kehityksen nykytilanteeseen. Kuvaan raportissa aluksi työn taustaa ja front-end-kehitystä yleisesti. Seuraavaksi käsittelen oppaan kirjoittamista, siihen laadittuja ohjeita, käytäntöjä sekä tekniikoita tarkemmin. Lopuksi kerron omat päätelmäni siitä, kuinka opas on otettu vastaan tiimissä, ja oppaan jatkokehitysmahdollisuudet. Opinnäytetyön lukijalta edellytetään tuntemusta WWW-kehityksen perusteista sekä HTML-, CSS- ja JavaScript-kielistä.

Työn tausta

Työskentelin alun perin Logica Suomi Oy:llä vuoden 2011 lokakuusta lähtien, josta siirryin toimeksiantajan palvelukseen syksyllä 2012 yritysoston myötä, kun CGI Group osti Logican. Kuulun CGI:llä Business Consulting -tiimiin. Työnkuvakseni on muodostunut, suureksi osaksi omasta halukkuudestani, sivustojen ja sovellusten front-end-kehitys. Opinnäytetyön idean sain tiimin WWW-suunnittelijalta, joka näki tarpeen kehittää tulevien sivustojen ja sovellusten front-end-puolta laatimalla yhteisiä käytäntöjä ja ohjeita yhteen paikkaan. Otin tehtävän vastaan, koska koin sen loistavana tilaisuutena kehittää omaa osaamistani ja

samalla parantaa jatkossa palveluidemme laatua. Aloitin työn samaan aikaan, kun liityin verkkokauppasovelluksemme uuden version kehitykseen alkukeväästä. Koimme sen olevan hyvä projekti käytäntöjen hyödyntämiselle ja määrittämiselle sitä mukaa kun tarpeita löytyy.

Toimeksiantaja

Opinnäytetyöni toimeksiantajana oli CGI Suomi Oy, tarkemmin sanottuna Tampereella toimiva Business Consulting -tiimi. Tiimi kehittää verkkopalveluita pääasiassa suomalaisille kaupan alan keskisuurille ja suurille yrityksille. Yritys on osa maailmanlaajuisesti toimivaa, vuonna 1976 perustettua IT-palveluyritys CGI Groupia. Yrityksen päätoimipiste on Montrealissa, Kanadassa. CGI:n palveluvalikoimaan kuuluvat liiketoiminta- ja IT-konsultointi, sovellusten kehitys ja hallinta, tietojärjestelmien väliset integraatiot, IT-infrastruktuuriratkaisut ja liiketoimintaprosesseihin liittyvät palvelut. Asiakaskunta koostuu rahoitus-, terveydenhuolto-, energia-, telekommunikaatio-, öljy- sekä kaupan alan yrityksistä ja julkisen sektorin organisaatioista. Henkilöstöä on 40 maassa n. 69 000, joista Suomessa työskentelee n. 3000 (Tampereella näistä n. 400). Vuonna 2012 konsernin liikevaihto oli n. 3,6 miljardia euroa. (CGI Group Inc 2013a.)

Tavoite ja tarkoitus

Työn tavoitteena oli yhtenäistää tulevissa projekteissa verkkosivustojen ja sovellusten front-end-koodia, parantaa ylläpidettävyyttä, nopeuttaa jatkokehitystä ja helpottaa uusien kehittäjien liittymistä tiimiin. Henkilökohtaiseksi tavoitteeksi otin tietojeni päivittämisen entuudestaan tuttujen tekniikoiden osalta sekä uusiin tutustumisen ja niiden opiskelun.

Tarkoituksena oli laatia yhteisiä käytäntöjä ja ohjeita ohjelmistokehittäjille suunnattuun, englanninkieliseen oppaaseen. Eri projekteja varten tarvitaan erilaisia käytäntöjä, joten toimeksiantaja halusi alkuun oppaan, jota voitaisiin hyödyntää kaikissa projekteissa. Siksi aiheiksi valittiin selaintuen määrittäminen, suorituskyvyn optimointi, esteettömyyden saavuttaminen ja front-end-koodin kirjoitta-

mista koskevat käytännöt. Opas haluttiin pitää tiiviinä ja kohderyhmälle soveltuvana, joten siitä rajattiin pois tarkat selitykset käytännöistä ja käytetyistä tekniikoista. Asiasta kiinnostuneille kirjattiin linkkejä resursseihin jatko-opiskelua varten. Lisäksi käyttöliittymän ulkoasuun liittyvät ohjeistukset ja käytännöt päätettiin jättää pois.

Lähteet

Tämän työn tekemistä varten päätin käyttää lähes poikkeuksetta verkkolähteitä. Keränen ja Penttinen (2007, 13–14) näkevät niiden ongelmana tiedon luotettavuuden arvioinnin. Sivustoilta puuttuu usein maininta tekstin kirjoittajasta tai julkaisupäivämäärästä. Puutteet vaikeuttavat kirjoittajan asiantuntemuksen arviointia ja voivat tarkoittaa sitä, että tieto on jo vanhentunutta. Sivustoja ei ole välttämättä päivitetty vuosiin, joten luotettavuuden arviointi jää omalle vastuulle. Keränen ja Penttinen (2007, 14) pitävät yritysten sekä organisaatioiden sivustoilla olevaa tietoa ja erilaisten tietopankkien sähköisiä julkaisuja luotettavina. Sen sijaan heidän mielestään markkinointitekstit, keskustelualueiden viestit ja blogikirjoitukset eivät ole objektiivisia.

Pyrin välttämään edellä mainittua ongelmaa valitsemalla lähteiksi sellaisten kirjoittajien julkaisuja, jotka ovat tunnettuja front-end-kehittäjäyhteisössä. He ovat henkilöitä, jotka puhuvat alan tapahtumissa, ja heillä on julkaistuja kirjallisia teoksia tai kirjoituksia on julkaistu arvostetuilla, ylläpidetyillä sivustoilla kuten Smashing Magazine (http://smashingmagazine.com) tai A List Apart (http://alistapart.com). Jos kirjoittaja tai aihe ei ollut itselleni aiemmin tuttu, tutkin tiedon luotettavuutta etsimällä viittauksia lähteeseen tai selvitin, mitä muissa teksteissä on kirjoitettu aiheesta. Haastetta lähteiden valintaan aiheutti niiden runsaus.

Työn hyödyt

Koen oppaasta olevan hyötyä kaikille, jotka työskentelevät projekteissa front-end-puolen parissa. Olen ollut CGI:llä työskennellessäni mukana kahdessa laajemmassa projektissa. Toisessa niistä olen yksin vastuussa front-end-puolesta, mutta olen huomannut, kuinka omatkin koodaustavat unohtuvat välillä, jos projektin parissa ei työskentele hetkeen. Tässä tapauksessa kirjatut käytännöt ja ohjeet auttavat hyvin palauttamaan ne mieleen. Aiemmin mainitsemassani verkkokauppaprojektissa minä, WWW-suunnittelija ja muut, enimmäkseen back-end-koodia kehittävät henkilöt toteuttavat front-end-puolta. Se koostuu kymmenistä dokumenteista, ja jokaisella on omat tapansa kirjoittaa koodia. Yhteiset pelisäännöt pitävät koodin ylläpidettävänä sekä luettavana ja helpottavat uusien toimintojen kehittämistä. Minun on pidettävä itseni ajan tasalla alan muutoksista ja kehityksestä, jotta opas pysyy hyödyllisenä. Oma osaamiseni karttuu samalla.

2 OPPAAN KIRJOITTAMINEN

Toimeksiantaja ei asettanut oppaalle erityisesti vaatimuksia, joten minulla oli hyvin vapaat kädet sen toteutukseen. Päädyin suunnittelemaan ja laatimaan sen yhteistyössä tiimin WWW-suunnittelijan kanssa, joka toimi samalla toimeksiantajan puolella opinnäytetyöni ohjaajana. Aloitin oppaan laatimisen samanaikaisesti verkkokauppa-sovelluksen uuden version kehityksen kanssa. Tämä tarkoitti sitä, että käytäntöjä voitiin määritellä sitä mukaa, kun tarpeita löytyi. Olin aiemmin kirjoittanut päivitys- ja ylläpito-oppaita freelancerina ja toiminimellä työskennellessäni. Aiemman kokemukseni lisäksi hyödynsin muita vastaavia oppaita ja WWW-suunnittelijan tietämystä.

Itselläni ja WWW-suunnittelijalla oli valmiiksi näkemystä siitä, millaisia aiheita oppaan tulisi käsitellä. Halusimme, että oppaassa on käytäntöjä ja ohjeistuksia liittyen front-end-kehityksen eri osa-alueisiin. Oppaan kohderyhmä, muut kehittäjät, vaikutti sisältöön ja rakenteeseen parilla tavalla. Tiesimme, etteivät kaikki kehittäjät ole yhtä kiinnostuneita aiheiden yksityiskohtaisia kuvauksista sekä valittujen käytäntöjen ja ohjeiden perusteluista. Tästä syystä ne rajattiin pois, jotta oppaasta tulisi tiivis ja sivumäärä pysyisi lyhyenä. Kirjasin kuitenkin linkkejä aiheisiin liittyviin resursseihin jatko-opiskelua varten. Tiimin kehittäjien osaamisalueet sijoittuvat enemmän back-end-kehitykseen, joten käytin havainnollistamiseen runsaasti lyhyitä koodiesimerkkejä ohjeiden tukena. Esimerkeistä on toki hyötyä niillekin, jotka tuntevat front-end-kehitystä paremmin.

Gimmebar (2013) listaa useita erilaisia front-end-tyylioppaita ja mallikirjastoja. Mallikirjastot (pattern library) ja yhdentyyppiset tyylioppaat ovat visuaalisia ohjeistoja HTML-elementtien sekä yleisimmin käytettyjen käyttöliittymäkomponenttien (valikot, painikkeet jne.) ulkoasusta ja merkkauksesta. Ajankäytön vuoksi rajasimme kuitenkin visuaalisuuteen ja ulkoasun toteuttamiseen keskittyvät käytännöt pois. Toisenlaiset tyylioppaat keskittyvät HTML-merkkauksen, CSS-tyylien ja standardeihin ja käytäntöihin. Esim. Google ja Yahoo ovat julkaisseet omat koodityylioppaansa. Näiden lisäksi löytyy laajempia kehitysoppaita (esim. Isobar Front-end Code Standards & Best Practices, http://isobar-idev.github.io/code-standards), joissa ohjeistetaan koodauksen lisäksi samoja asioita joita halusimme oppaaseen. Valitsin Isobarin oppaan omani pohjaksi.

Oppaassa on toistaiseksi vain yleisiä käytäntöjä, joita voidaan soveltaa kaikissa projekteissa. Jatkossa kuitenkin voi olla tarpeen laatia tarkempaa ohjeistusta tiettyjen sivustojen ja sovellusten koodauksesta. Oppaan ensimmäinen versio on julkaistu dokumenttina, mutta laajentamisen ja luettavuuden kannalta parempi esittämistapa olisi HTML-muotoinen dokumentaatio. Toteuttamiseen kannattaa mielestäni käyttää valmista työkalua (esim. Daux.io, http://daux.io), ettei aikaa kulu turhaan dokumentaation pohjan koodaamiseen.

2.1 Julkaisumuoto

Dokumentin tiedostomuotoon tulee kiinnittää huomiota. Dokumentit eivät välttämättä aukea muilla toimisto-ohjelmistoilla, kuin sillä, jonka tiedostomuodossa dokumentti on tallennettu. Toimisto-ohjelmistot käyttävät ohjelmistokohtaisia tiedostomuotoja. Esimerkiksi OpenOfficessa voi avata Microsoft Officen tiedostoja, mutta asiakirjojen tyylit voivat muuttua. Julkaisua ja jakelua varten voidaan käyttää myös kyseisiin tarkoituksiin paremmin soveltuvaa muotoa kuten PDF:ää, joissa huomioidaan internet-jakelun tarpeet sekä dokumentin helppo käytettävyys. PDF:nä tallennettuna dokumentit ovat pienempiä tiedostokooltaan kuin toimisto-ohjelmistojen muodoissa tallennetut. Käytetty fontti on sisällytettynä PDF-dokumenttiin, joten dokumentti näyttää aina samalta ympäristöstä huolimatta (Keränen, Penttinen 2007, 160). Opas päätettiin kuitenkin PDF:n hyödyistä huolimatta julkaista Microsoft Wordin DOCX-muotoisena, koska sen on tarkoitus olla muiden kehittäjien muokattavissa. Yrityksen työasemiin ja kannettaviin tietokoneisiin on oletuksena asennettu Microsoft Office -ohjelmisto.

2.2 Teksti

Oppaan peruselementtinä on teksti, jonka ensisijainen tarkoitus on viestinnällinen. Sillä kerrotaan asioita, välitetään tietoa sekä herätetään mielikuvia ja tuntemuksia. Sisällön lisäksi kirjoitustyyli vaikuttaa lukijaan, ja sama asia voidaan kirjoittaa eri tavoilla riippuen siitä, millainen vaikutus halutaan tehdä. Näiden lisäksi tekstillä on graafisena elementtinä visuaalinen merkitys, johon kiinnitetään huomiota typografiassa ja asemoinnissa. Typografia tarkoittaa yksittäisten kirjainten ulkoasua, asemointi on taas kokonaisten tekstien ja kuvien asettelua sivulla. (Keränen, Penttinen 2007, 170.)

Dokumenttia voidaan lukea joko näytöltä tai paperilta, joten tapojen erot on otettava huomioon tekstissä. Hitaamman lukemisen lisäksi näytöllä katse harhailee, joten luettua on vaikeampi hahmottaa kokonaisuutena. Syinä tähän voivat olla erilaiset keskittymiseen vaikuttavat häiriötekijät, kuten monitorin etäisyys, heijastukset ja tekstin vierittäminen. Myös tekstin muistaminen ja ymmärtäminen on vaikeampaa. Lukemista voidaan helpottaa muutamilla tekijöillä. Lyhyt ja ytimekäs teksti on helpommin hahmotettavaa kuin pitkät, usealle riville jakautuneet kappaleet. Kappaleiden tulisi olla alle 10 riviä pitkiä ja selkeästi otsikoitua. Väliotsikoiden tarkoituksena on tiivistää tulevan tekstin sisältö. Hyvin tehtynä ne auttavat lukijaa silmäilemään tekstiä, löytämään halutun kohdan dokumentista ja herättävät kiinnostusta sisällöstä. (Keränen, Penttinen 2007, 170–171.)

3 FRONT-END-KEHITYS

Sivuston tai sovelluksen front-end-puoli, suomeksi ilmaistuna käyttöliittymä tai edustajärjestelmä huolehtii käyttäjälle näytettävästä ulkoasusta, vuorovaikutuksesta käyttäjän kanssa ja päätelaitteessa toimivasta logiikasta sekä käsiteltävästä datasta (Korpela & Lehdonvirta 2013, 15). Sivuston tai sovelluksen taustajärjestelmä, back-end, koostuu yleensä kolmesta osasta: palvelimesta, esim. PHP:lla, Javalla tai C#-kielellä ohjelmoidusta sovelluksesta ja tietokannasta (Long 2012). Kärjistän selkeyden vuoksi verkkosivuston ja -sovelluksen eroa. Verkkosivusto on kokoelma toisiinsa linkitettyjä dokumentteja, jonka sisältö ei muutu käyttäjän toimien perusteella. Verkkosovelluksen toiminta ja sisältö sen sijaan perustuvat käyttäjän tekemiin toimiin. Todellisuudessa erottelu ei ole täysin mustavalkoista, sillä nykypäivänä sivustot saattavat sisältää useita sovellusmaisia osia.

Front-end-kehityksellä tarkoitetaan tässä työssä nimenomaan verkkosivustojen ja -sovellusten käyttöliittymien kehitystä, eikä esim. mobiili- tai työpöytäsovellusten. Lähdemateriaalia ja alan työpaikkailmoituksia tutkiessani olen todennut tämän olevan nykyään vakiintunut merkitys. Kehitys perustuu Korpelan ja Lehdonvirran (2013, 15) kuvaamien tarkoitusten täyttämiseen HTML-merkkauksella, CSS-tyyleillä ja JavaScript-koodin avulla. Olen havainnut, että alaa vähemmän tuntevat saattavat nähdä sen pelkästään tyylikkään ulkoasun toteuttamisena. Vaikka ulkoasu on myyvänä tekijänä sivustossa tai sovelluksessa oleellista, lisäksi siihen kuuluu monia muita tekniikoita ja käsitteitä, joista tässä työssä käsiteltäviä ovat mm. selaintuki, suorituskyky ja esteettömyys.

Mobile-first-lähtökohta ja mukautuva suunnittelu

Internetin käyttö älypuhelimilla, tableteilla ja muilla laitteilla kuin tietokoneilla (esim. televisiot) kasvaa jatkuvasti. Videopalvelu Netflixiä käytetään lähes tuhannella erilaisella Android-laitteella (Netflix Inc 2012) ja tutkimusfirma Gartner Inc:n (2011) mukaan tabletit ja älypuhelimet muodostavat 90 % uusien laitteiden hankinnassa tapahtuvasta kasvusta vuoteen 2015 mennessä. Uskon, että markkinoille tulee jatkossa laitteita, joita käytetään täysin eri tavoilla kuin nykyi-

siä puhelimia tai tabletteja. Esimerkki tällaisista laitteista on Googlen Glass (http://www.google.com/glass). Ne vaativat taas osaltaan uusia lähestymistapoja front-end-kehitykseen. Viime vuosina kasvanut internetin käyttö mobiililaitteilla ilmenee kuvasta 1.

Mobile share of web traffic			
	2010	2012	Increase 2010-2012
Africa	5.81%	14.85%	155.59%
Asia	**6.1%**	**17.84%**	**192.46%**
Europe	1.81%	5.13%	183.43%
North America	4.71%	7.96%	69.00%
Oceania	2.88%	7.55%	162.15%
South America	1.46%	2.86%	95.89%
Worldwide	3.81%	10.01%	162.73%

KUVA 1. Mobiilikäytön osuuden kasvu internet-liikenteessä vuodesta 2010 vuoteen 2012 (Smart Insights 2013)

Kuisman (2012) mielestä edellä mainitun muutoksen vuoksi verkkopalvelujen suunnittelun on sopeuduttava laitteiden ja käyttötapojen monipuolistuvaan valikoimaan. Suunnittelun lähtökohtaa on muutettava, koska enää ei voida olettaa palveluiden käytön rajoittuvan työasemien tai kannettavien äärelle. Toimeksiantaja haluaa tehdä front-end-kehitystä mobile-first-lähtökohtaisesti ja hyödyntää mukautuvaa suunnittelua ratkaisemaan alan uusia haasteita. Päätelaitteiden monimuotoisuus, erilaiset käyttötavat (esim. hiirellä selaamisen eroavaisuus sormilla selaamiseen) ja rajoitukset mobiiliyhteyksien nopeudessa (kesämökillä tai teillä liikkuessa yhteyden nopeus ei aina ole paras mahdollinen) vaativat muutoksia kehitystapoihin.

Ethan Marcotten tunnetuksi tekemä mukautuva suunnittelu (responsive design) tarkoittaa käyttöliittymien kykyä sopeutua käytetyn laitteen ominaisuuksien perusteella. Kyseinen suunnittelutapa esiteltiin alun perin Marcotten (2010) artikkelissa ”Responsive Web Design”. Yksinkertaisimmillaan se voi näkyä esimerkiksi niin, että ruudun koko vaikuttaa sivuston tai sovelluksen asetteluun. Puhelimella selattuna sisältö voi olla yhdessä palstassa, kun se onkin tabletilla jaettuna kahteen palstaan. Mukautuvuus perustuu CSS:n media-kyselyihin (media

query), joiden avulla voidaan määritellä käyttöliittymän muuttumista laitteiden eri ominaisuuksien perusteella. Media-kyselyitä voidaan käyttää kaikilla moderneilla selaimilla. Tällä hetkellä yleisin käyttötapaus on asettelun pohjustaminen selainikkunan leveyteen. (Kuisma 2012.)

Mukautuvat käyttöliittymät ovat osa mobile-first-lähtökohtaa. Kuisman (2012) mukaan suunnittelustrategian ajatuksena on toteuttaa käyttöliittymät ”heikoimman yhteisen nimttäjän” mukaan, vaiheittaisella parantamisella (progressive enhancement). Mobiililaitteet eivät ole suorituskyvyltään tai ominaisuuksiltaan työasemien veroisia, joka saattaa aiheuttaa ongelmia, jos isommille näytöille tarkoitettuja käyttöliittymiä skaalataan pienemmäksi puhelimia ja tabletteja varten. Siksi on usein kannattavampaa rakentaa käyttöliittymät näiden ehdoilla ja laajentaa front-end-puolta työpöytäselaimia varten. Kuisma (2012) näkee kuitenkin mukautuvan suunnittelun olevan vielä rajoittunutta ja se jää usein pelkästään asettelun ja sisällön uudelleenjärjestelyyn. Mielestäni sen lisäksi sivuston toiminnot, kuten navigaatiovalikot ja lomakkeet, sekä sisällön laatu (esim. korkearesoluutioisia ja tiedostokooltaan suuria kuvia on tarpeetonta ladata puhelimilla) tulisi mukauttaa laitteille.

Asiakkaat vaativat jo nyt sivustoihin ja sovelluksiinsa mobiililaitteille optimoituja käyttöliittymiä, joka tekee mukautuvuuden ja mobile-first-lähtökohdan käyttämisestä selkeän ja kustannustehokkaan ratkaisun toimeksiantajalle. Sivustolla tai sovelluksella on näin yksi koodipohja, eikä Android-, iOS- ja Windows-laitteille tarvitse kehittää erillisiä ratkaisuja, puhumattakaan muista tulevaisuudessa mahdollisesti markkinoilla olevista käyttöjärjestelmistä. Mobiilisovellusten kehittäminen vaatii oman osaamisensa, jota harvalla tiimin kehittäjällä on, kun taas kaikki osaavat vähintään perusteet front-end-kehityksen kielistä. Myös Google (2012) suosittelee kehittäjille mukautuvaa suunnittelua.

4 OPPAAN KÄYTÄNNÖT

Oppaan käytännöt ja ohjeet on koostettu vastaavien oppaiden sisällöstä, omista kokemuksista sekä näkemyksistä ja projektien aikana vastaan tulleista hyvistä toimintatavoista. Niillä pyritään täyttämään luvussa 1 opinnäytetyölle asetettuja tavoitteita. Pidän olennaisimpina tavoitteina koodin yhtenäisyyttä ja luettavuutta. Front-end-koodia kirjoittavat yleensä useat kehittäjät, joista jokaisella on oma koodaustyylinsä. Tässä tapauksessa luettavuudella tarkoitetaan sitä, että koodin kirjoittaja ja varsinkin muut samaa koodia käsittelevät ymmärtävät sen toiminnan ja tarkoituksen tulevaisuudessa. Muutamissa lähteenä toimivissa oppaissa on tiivistetty käytäntöjen ”kultainen sääntö”: koko front-end-koodipohjan tulee näyttää yhden henkilön, ei tiimin, kirjoittamalta.

Tarkoitus ei ole kuitenkaan noudattaa ohjeita soveltamatta niitä projektin mukaan. Sivuston tai sovelluksen laajuus on tärkeä tekijä käytäntöjen noudattamisessa. Pienit projektit (esim. muutaman sivun markkinointisivusto) vaativat usein täysin erilaisia toiminta- ja kehitystapoja kuin laajemmat (esim. verkkokauppa ja sen ylläpitoliittymä). Aikataulut voivat olla tiukkoja, jolloin ei ole järkevää uhrata koko kehitysaikaa jokaisen tekniikan käyttöönottoon. Jos sivustossa tai sovelluksessa noudatetaan jo aiemmin sovittua koodaustyyliä, sitä ei pitäisi mielestäni korvata oppaan tyylillä paitsi erikseen päätettäessä. Tuntemus käytännöistä ja syistä niiden käyttöön on hyödyksi, koska tänä päivänä pätevät käytännöt saattavat vanhentua tai muuttua kokonaan turhiksi uusien tekniikoiden, työkalujen tai selainten kehityksen myötä.

4.1 Kolmannen osapuolen ratkaisut

Oppaaseen on listattu tärkeimpiä kolmannen osapuolen ratkaisuja, joita toimeksiantaja käyttää projekteissa. Ne ovat useimmiten JavaScript-kirjastoja, joilla toteutetaan esim. erilaisia käyttöliittymäkomponentteja tai hoidetaan back-end-puolelta tuotavan tiedon näyttämistä. Olen todennut, että näiden toimintojen toteuttaminen täysin tyhjästä vie huomattavasti aikaa, eikä sitä ole projekteissa käytettävissä loputtomasti. Siksi kehityksen nopeuttamiseksi pyörää ei aina kannata lähteä keksimään uudestaan, vaan on järkevämpää käyttää muiden

tekemiä ratkaisuja. Kuitenkin usein vaihtoehtoja löytyy jopa liian monia, esimerkkinä vaikka kuvakarusellin toteuttaminen.

GitHub (http://github.com) on sivusto, jossa kehittäjät voivat jakaa tuotoksiaan muiden muokattaviksi ja ladattaviksi. Sivustolla listataan yli 100 tulosta hakusanoilla ”image carousel”. Joskus selkeästi parasta vaihtoehtoa ei ole, vaan niiden soveltuvuutta täytyy arvioida jollain tapaa. Itse suosin kattavasti dokumentoituja ja GitHubissa suosittuja ratkaisuja. Kehittäjien ja yhteisön aktiivisuus kertovat paljon vikakorjausten ja päivitysten nopeudesta. Laadukas lähdekoodi ja sille kirjoitetut testit tuovat luotettavuutta. Lisäksi koodia tulisi pystyä räätälöimään ja laajentamaan muutenkin kuin lähdekoodia muokkaamalla. Se on riskialtista, koska kirjaston päivitykset voivat rikkoa omat muutokset.

Kirjaston tiedostokoko, suorituskyky (monimutkainen koodi saattaa hidastaa toimintojen suorittamista) ja selaintuki kannattaa ottaa huomioon valinnassa. Käyttöliittymäkomponenttien kohdalla myös esteettömyys (voidaan käyttää myös näppäimistöllä) ja mukautuvuus (tukee kosketusnäytöllä käyttämistä) ovat olennaisia seikkoja, jos ratkaisu on tarkoitettu asiakaskäyttöön tulevalle sivustolle tai sovellukselle. Komponenttien ulkoasua on usein muokattava omaan sivustoon tai sovellukseen sopivaksi. Merkkaustapa ja valmiit CSS-tyylit vaikuttavat siihen, kuinka vaivatonta muokkaus on.

4.2 Selaintuki

Ammattimaisen WWW-kehityksen ensimmäisten 10 vuoden aikana, 90-luvulla, selaintuki tarkoitti sitä, että sivusto joko toimi tai ei toiminut ollenkaan tietyllä selaimella. Usein pääsy sivustolle estettiin täysin, jos käytössä oli tukematon selain (Yahoo! Inc 2013). En muista itse nähneeni sellaisia tapauksia enää 90-luvun lopussa, kun itse tutustuin Internetiin. Sen sijaan monella sivustolla oli ilmoitus: ”Toimii parhaiten selaimella x ja resoluutiolla y.” Olen todennut, että modernit selaimet tukevat WWW:n standardeja jo niin hyvin, ettei kokonaisia käyttäjäryhmiä tarvitse sulkea pois sivustoilta tai sovelluksista. Mielestäni front-end-kehitystä ei tarvitse enää tehdä yksittäisen selaimen ehdoilla, ellei se ole tarpeen sivuston tai sovelluksen kohderyhmän takia (esim. organisaatioissa ei

ole välttämättä mahdollista käyttää muuta selainta kuin vanhaa Internet Explorerin versiota).

Selaintuki ei tarkoita sitä, että jokaiselle käyttäjälle taataan täysin sama käyttökokemus. Sivustojen ei tarvitse näyttää samalta kaikissa selaimissa, eikä kaikkien vähemmän kriittisten toimintojen tarvitse olla käytettävissä vanhemmilla selaimilla. Pikselintarkkaa ulkoasua ja täydellistä toimintaa tärkeämpää on varmistaa, että kaikki oleellinen sisältö on jokaisen käyttäjän saatavilla. Sen saavuttamiseksi voi hyödyntää vaiheittaista parantamista. Modernien selaimien käyttäjille on hyväksyttävää tarjota näyttävämpi ja uusia tekniikoita hyödyntävä käyttöliittymä. Ennen uusien tekniikoiden käyttämistä kannattaa kuitenkin tutkia, millä selaimilla ja versioilla ne toimivat. Sivustot, kuten HTML5 Please (http://html5please.com) ja Can I Use (http://caniuse.com), tarjoavat tietoa tekniikoiden selaintuesta.

Näkemykseni mukaan nykyään mahdollisia päätelaitteiden, resoluutioiden, selainten ja niiden eri versioiden yhdistelmiä on niin monia, ja projekteissa kehitysaika on rajallista, ettei jokaista variaatiota ehdi tai pysty millään varmistamaan erikseen. Se ei olisi järkevää tai kustannustehokasta, vaikka aika riittäisikin. Mielestäni mukautuvaa suunnittelua sekä vaiheittaista parantamista hyödyntämällä ja mobile-first-lähtökohdalla päästään jo pitkälle kattavan selaintuen varmistamisessa. Oppaaseen on määritelty selaimet, joissa sivustoille ja sovelluksille taataan täysi, varmistettu toimivuus. Valinnat perustuvat tutkittuihin selainten käyttöosuuksiin, joista nähdään tämän hetken suosituimmat selaimet. Selaimet on listattu taulukossa 1.

TAULUKKO 1. Viisi suosituinta selainta Suomessa vuonna 2013 (StatCounter Global Stats 2013)

Selain	**Versio**	**Prosenttiosuus**
Chrome	Kaikki	34,5 %
Firefox	5+	34,06 %
Internet Explorer	9.0	7,39 %
Internet Explorer	8.0	5,05 %
Internet Explorer	10.0	4,67 %

4.3 Suorituskyky

Sivustot ja sovellukset vaativat usein suuren määrän muuttuvaa sisältöä käyttöliittymissä sekä monia resurssitiedostoja, jotka lisäävät latausnopeutta ja vaikuttavat suorituskykyyn. Googlen tutkimuksen (Web Site Optimization 2012) mukaan puolen sekunnin lisäys hakutuloksien näyttämisessä voi vähentää liikennettä ja mainostuloja 20 prosenttia. Käyttäjät odottavat sivujen latautuvan kahdessa sekunnissa, ja kolmen jälkeen 40 prosenttia käyttäjistä poistuu sivustolta (Swanson 2013). Esim. verkkokaupan kohdalla ei ole suotavaa, että käyttäjä ehtii turhautua ja poistua ennen kuin ostotapahtuma tehdään loppuun. Myös sijoittuminen Googlen hakutuloksissa saattaa kärsiä, koska niissä otetaan huomioon nopeus yhtenä seikkana (Search Engine Land 2010).

Tomlinsonin (2012) mukaan kolme tekniikkaa muodostavat hyvän pohjan suorituskyvyn optimoinnille. Ne ovat yhteenliittäminen (concatenation), pakkaus (compression) ja selaimen välimuistin hyödyntäminen (caching). Kyseisillä tekniikoilla vähennetään selaimen tekemien HTTP-pyyntöjen määrää ja pienennetään ladattavien resurssien kokoa. Tämä näkyy sivustossa tai sovelluksessa lyhempänä latausaikana. Jokaiselle löytyy erilaisia työkaluja: työpöytä- sekä verkkosovelluksia ja ohjelmointikielien liitännäisiä, jotka ajetaan palvelimella sivuston tai sovelluksen käännösprosessin yhteydessä. Täydellinen optimointi vaatii kuitenkin edellä mainittujen lisäksi muitakin toimenpiteitä. Selainvalmistajien sivustot tarjoavat kattavan katsauksen suorituskykyyn liittyvistä käytännöistä, jotka löytyvät osoitteista http://developer.yahoo.com/performance/rules.html ja http://developers.google.com/speed/docs/best-practices/rules_intro.

Yhteenliittäminen

Tomlinsonin (2012) mukaan yhteenliittämisellä tarkoitetaan useiden samantyyppisten resurssitiedostojen (kuvat sekä JavaScript- ja CSS-dokumentit) yhdistämistä yhdeksi tiedostoksi. Tarkoituksena on vähentää palvelimelle tehtäviä HTTP-pyyntöjä, joita varten tarvittavien yhteyksien muodostaminen voi kestää kauemmin kuin itse resurssien siirtäminen. Jokainen pyyntö pidentää sivun latausaikaa, koska selaimet käsittelevät niitä vain rajallisen määrän samanaikai-

sesti. Se on huomattavissa esim. mobiililaitteilla hitaamman yhteyden varassa selatessa, kun kuvat latautuvat yksi kerrallaan (Tomlinson 2012).

Tomlinson (2012) suosittelee varsinkin JavaScript-dokumenttien yhdistämistä, koska selaimet keskeyttävät sivun piirtämisen siksi aikaa, kunnes dokumentit on ladattu ja käsitelty. Vaikka CSS-dokumenttien kanssa ei ole samaa ongelmaa, kannattaa nekin silti yhdistää. Yhdistäminen tulisi tehdä vasta sivuston tai sovelluksen siirtyessä tuotantoon. Olen huomannut, että kehitysvaiheessa ylläpito hankaloituu ja luettavuus kärsii jos useaan dokumenttiin jaettu koodi yhdistetään. Koodiesimerkissä 1 nähdään lyhyt esimerkki yhteenliitetyistä JavaScript-dokumenteista.

HTML:

```
<script src="jquery.js">
<script src="jquery-ui.js">
<script src="shopping-cart.js">
<script src="login.js">
<script src="main.js">
```

→

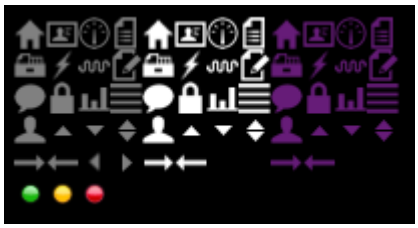
```
<script src="app.js">
```

JavaScript, app.js:

```
*jQueryn koodi*
*jQuery UI:n koodi*
…
// shopping-cart.js
*Omaa koodia*
…
// login.js
*Omaa koodia*
// main.js
*Omaa koodia*
```

KOODIESIMERKKI 1. Useampi JavaScript-dokumentti yhteenliitettynä

Käytän usein paljon pieniä ikoneita visuaalisuuden lisäämiseksi sivustoilla ja sovelluksissa. Jokaista ikonia varten ei ole järkevää tehdä erillistä HTTP-pyyntöä, vaan kuvat voidaan yhdistää yhdeksi isommaksi kuvaksi, spriteksi. Tarvittava ikoni näytetään HTML-elementin taustakuvana, oikeasta kohtaa spriteä CSS-tyylillä. Spriteen voi toki yhdistää muutakin grafiikkaa, esim. sivuston taustana toimivan kuvion tai logon. Vaikka tekniikka on hyvä suorituskyvyn kannalta, ylläpito vaikeutuu, koska kuvan muuttuessa myös tyylejä on muutettava. Kuvassa 2 on toimeksiantajan toteuttama image sprite, jossa on yhdistetty useita ikoneita.



KUVA 2. Image sprite (CGI Group Inc 2013b)

Pakkaus

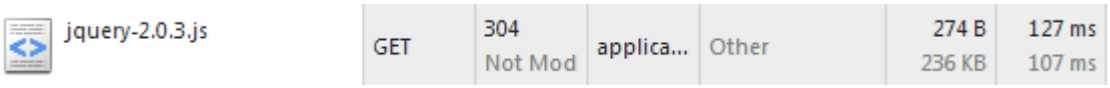
Suorituskyvyn parantamisessa päästään hyvään alkuun yhteenliittämisellä, mutta muita toimenpiteitä voidaan vielä tehdä. Lähes kaikki palvelimet ja selaimet tukevat HTTP-pakkausta, jonka suosituimmat menetelmät ovat gzip ja deflate. Molemmat pienentävät resurssien kokoa jo palvelimen päässä, ennen kuin niitä ladataan käyttäjän selaimeen. (Tomlinson 2012.)

Lisäksi palvelimelta haettavan tiedon määrää voidaan vähentää dokumenttien minifikaation sekä pakkauksen ja kuvien optimoinnin avulla. Minifikaatio poistaa JavaScript- ja CSS-dokumenteista selaimelle turhaa dataa, kuten kommentit ja tyhjän tilan. JavaScript-koodin kohdalla prosessi on monimutkaisempi. Jotkut työkalut lyhentävät muuttujien nimiä yksimerkkisiksi ja korvaavat lausekkeita lyhemmillä vastaavilla. Kuvissa on usein dataa, joka on poistettavissa ilman kuvalaadun heikkenemistä. Jos laadun heikkeneminen ei ole huolenaihe, esim. jos kuva näytetään pienenä sivustolla tai sovelluksessa, sen voi tallentaa uudelleen pienemmäksi pakattuna. (Tomlinson 2012.)

Koodiesimerkissä 2 ja kuvassa 3 on esimerkit pakkaamattomasta koodista ja dokumentin koosta. Koodiesimerkissä 3 on sama koodi pakattuna. Kuvassa 4 nähdään kuinka pakkaus vaikuttaa tiedostokokoon.

```
// jquery.js
jQuery.fn = jQuery.prototype = {
        // The current version of jQuery being used
        jquery: core_version,

        constructor: jQuery,
…
```

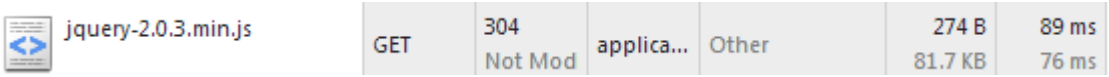
KOODIESIMERKKI 2. JQuery-kirjaston koodia



KUVA 3. Chrome-selaimella ladattu täysi kehitysversio jQuery-kirjastosta. Tiedostokoko 236 kb.

```
// jquery.min.js
x.fn=x.prototype={jquery:p,constructor:x,
…
```

KOODIESIMERKKI 3. JQuery-kirjaston koodia pakattuna



KUVA 4. JQuery-kirjasto pakattuna. Tiedostokoko 81.7 kb.

Välimuistin hyödyntäminen

Sivustolle tai sovellukseen palaavia käyttäjiä voidaan auttaa hyödyntämällä selaimen välimuistia. Oletuksena selain joutuu lataamaan kaikki resurssitiedostot uudelleen. HTTP-protokollassa on kaksi laajalti käytössä olevaa tekniikkaa, joilla uudelleenlataukseen voi vaikuttaa: cache-otsikot (cache headers) ja entity-tagit (ETags). Ne voidaan määritellä back-end-koodissa, palvelimen konfiguraatiotiedostoissa tai cache-otsikoiden kohdalla merkkauksessa meta-tageilla. (Tomlinson 2012.)

Cache-otsikot soveltuvat käytettäväksi harvoin tai ei koskaan muuttuvien resurssien kanssa. Otsikoiden Expires-arvo määrittää päivämäärän, jonka jälkeen resurssi on ladattava uudestaan. Cache-Control: maxage kertoo sekuntimäärän, kuinka kauan resurssia voidaan käyttää selaimen välimuistista. ETag on tarkoitettu jatkuvasti muuttuvalle resursseille. Se on eräänlainen versiotunniste, jolla tarkistetaan, onko välimuistissa oleva resurssi sama kuin palvelimella oleva. ETagit eivät ole yhtä tehokas ratkaisu kuin cache-otsikoiden käyttö, koska ne vaativat kaikissa tapauksissa kyselyn palvelimelta. (Tomlinson 2012.)

Ajan perusteella tehtävässä uudelleenlatauksessa on myös huono puolensa. Jos resurssi muuttuu ennen ajan umpeutumista, eivät käyttäjät näe muutosta (Tomlinson 2012). Omalla kohdallani tämä on tullut projekteissa useasti vastaan tyylejä ja JavaScriptiä muokatessa, kun toinen kehittäjä ei näekään omassa selaimessaan muutoksia. Ongelma voidaan estää cache-busting-tekniikalla. Muuttuvan versiotunnisteen lisääminen resurssin URL-osoitteeseen aiheuttaa resurssin uudelleenlatauksen aina tunnisteen vaihtuessa (Tomlinson 2012). Ylläpidon kannalta tunnisteiden luominen kannattaa automatisoida back-end-puolen tekniikoilla. Koodiesimerkissä 4 kuvataan yksi tapa versiotunnisteen merkkaamiseen.

```
http://example.com/style.css?version=123

→

http://example.com/style.css?version=124
```

KOODIESIMERKKI 4. CSS-dokumentti versioituna ja päivityksen jälkeen

4.4 Esteettömyys

WWW on suunniteltu kaikkien käytettäväksi huolimatta käyttäjän päätelaitteesta ja selaimesta, kielestä, kulttuurista, sijainnista ja fyysisestä tai henkisistä kyvyistä. Huonosti suunnitellut sivustot ja sovellukset estävät jonkin rajoitteen tai vamman omaavia käyttäjiä suorittamasta toimintoja. Esimerkiksi hiiren käyttöön kykenemättömät eivät pysty suorittamaan ostoksia loppuun verkkokaupassa,

jos toiminnot eivät tue käyttöä näppäimistöllä. Näkökyvyltään rajoittuneet eivät pääse käsiksi pelkästään kuvina (esim. kuvaajat) esitettävään sisältöön, jos niille ei ole määritelty tekstivastineita. Esteettömyys sivustoilla ja sovelluksissa tarkoittaa sitä, että kaikki käyttäjät voivat havaita ja ymmärtää sisällön, navigoida sekä toimia niissä. Vaikka esteettömyydellä autetaan ensisijaisesti vammautuneita käyttäjiä, siitä hyötyvät myös esim. mobiililaitteilla selaavat ja huonon verkkoyhteyden varassa olevat (Web Platform Docs 2013).

Verkon esteettömyyttä pidetään maailmanlaajuisesti keskeisenä tavoitteena. YK tunnistaa sen kuuluvan perusihmisoikeuksiin (Web Platform Docs 2013) ja Yhdysvalloissa velvoitetaan laissa julkisen sektorin sivustojen sekä sovellusten noudattavan 16 kohdan ohjeistoa esteettömyydestä (WebAIM 2013). Section 508 -tarkistuslista löytyy osoitteesta http://webaim.org/standards/508/checklist. Suomessa on sivustoja ja sovelluksia koskeva julkisen hallinnon suositus, mutta ei toistaiseksi varsinaisia standardeja eikä lainsäädäntöä (Essityöryhmä 2003).

Front-end-koodin kirjoittamisen tukena kannattaa käyttää esim. W3C:n (World Wide Web Consortium, kansainvälisiä verkkostandardeja ylläpitävä organisaatio) laatimia ohjeistoja, joihin edellä mainittu Section 508 -tarkistuslista kuuluu. Mielestäni yksi parhaista ohjeistoista löytyy Web Accessibility -sivustolta (https://www.webaccessibility.com/best_practices.php), jota pidän luettavampana ja selkeämpänä kuin W3C:n ylläpitämiä listoja. Merkkauksen laadun voi tarkistaa myös automaattisesti työkaluilla kuten WAVE (http://wave.webaim.org). Näkemykseni mukaan esteettömyyden saavuttamiseksi pelkkä hyvä koodi ei riitä, vaan se otettava huomioon myös ulkoasun ja sisällön suunnittelussa.

5 OPPAAN TEKNIIKAT

Kuten luvussa 4 kerrottiin, kolmannen osapuolen ratkaisujen käyttämisellä voidaan säästää runsaasti kehitysaikaa. Tässä luvussa esitellään tarkemmin kaksi tekniikkaa, front-end-sovelluskehykset ja esikäsittelijät. Ne eivät liity pelkästään yhteen kieleen, vaan jokaiselle on saatavilla omia ratkaisujaan. Eri kieliä käsittelevissä luvuissa käsitellään pelkästään niitä koskevia tekniikoita.

5.1 Front-end-sovelluskehykset

Sovelluskehyksen tarkoituksena on nopeuttaa uusien sovelluksien kehittämistä tarjoamalla valikoiman valmiita toimintoja liittyen mm. käyttäjien hallintaan, tiedon tallennukseen ja verkkosovelluskehyksissä sessionhallintaan. Työpöytäkehykset taas saattavat sisältää yleisiä käyttöliittymäkomponentteja. Niiden käyttö vähentää perustoiminnallisuuksien ja rakenteen toteutukseen kuluvaa aikaa, jolloin kehittävät voivat keskittyä sovelluskohtaisten vaatimusten toteuttamiseen. Sovelluskehysten koodi on kattavasti testattua ja muiden kehittäjien käyttämää, joka pienentää virheiden esiintymisen mahdollisuutta. Front-end-sovelluskehykset kattavat samoja tarpeita, tai voivat vaihdella ulkoasun asettelua helpottavasta ruudukosta (grid, esim. http://semantic.gs) tiettyä kieltä, esim. JavaScriptiä mittavasti laajentavaan työkaluvalikoimaan (esim. jQuery ja MooTools).

Kuten muidenkin kolmannen osapuolen ratkaisujen kanssa, olen huomannut, että front-end-sovelluskehyksiä on nykyään runsaasti erilaisia. Vaikuttaa lähes siltä, että niitä kehitetään pelkästään tekemisen ilosta, eikä välttämättä oikeaan tarpeeseen. Valinta kannattaa siis tehdä luvussa 2 listaamieni periaatteiden mukaan (dokumentoinnin määrä ja laatu, suosio ja aktiivisuus kehittäjäyhteisössä, koodin laatu).

Twitter Bootstrap

Twitterin kehittäjä Mark Otto (2012) kuvaa Bootstrapia avoimen lähdekoodin front-end-työkaluvalikoimaksi, jonka tarkoituksena on auttaa suunnittelijoita ja kehittäjiä rakentamaan sivustoja sekä sovelluksia tehokkaasti ja nopeasti. Bootstrap sai alkunsa vuonna 2010 Oton ja toisen kehittäjän, Jacob Thorntonin toimesta. He halusivat tarjota Twitterin sisäiseen ja muiden vapaaseen käyttöön pitkälle jalostetun, hyvin dokumentoidun ja kattavan kirjaston räätälöitäviä käyttöliittymäkomponentteja (Otto 2012).

Bootstrapin perustan muodostavat selainkohtaisten erojen normalisointiin, ulkoasun asetteluun ja HTML-elementtien muotoiluun liittyvät tyylit. Pidän sen hyödyllisimpänä osana Oton (2012) kuvaamia räätälöitäviä komponentteja. Ne ovat käyttöliittymissä yleisimmin käytettyjä toimintoja, kuten pudotus- sekä navigaatiovalikoita ja painikkeita. Komponenttien toiminnot on toteutettu JavaScriptillä. Puutteitakin toki löytyy, eikä kaikkia vakiintuneita komponentteja ole vielä mukana. Itse kaipaisin tuleviin versioihin mukaan päivämäärä- (datepicker) ja liukuvalitsimen (slider). Ne ovat löydettävissä toisista front-end-sovelluskehyksistä ja erillisistä kirjastoista, mutta ylläpidon kannalta välttäisin mieluummin toisen kehyksen tai ylimääräisten kirjastojen käyttöönottoa pelkästään muutaman komponentin takia.

Kirjoitushetkellä Bootstrapista on julkaistu versio 3, mutta toimeksiantaja käytti sitä jo ennen virallista julkaisua. Se sopii erinomaisesti projektien pohjaksi, koska kehys on suunniteltu mobile-first-lähtökohdan mukaisesti.

5.2 Esikäsittelijät

Merkkauksen ja CSS-tyylien kirjoittaminen on osoittautunut itselleni varsin työlääksi ja aikaa vieväksi jo pienempien harrasteprojektien kohdalla, puhumattakaan työelämässä toteutetuista laajemmista sivustoista ja sovelluksista. Merkkauksessa on oltava tarkkana syntaksin kanssa, kun ominaisuudet täytyy ympäröidä lainausmerkeillä ja tagit on aloitettava sekä suljettava oikein. CSS taas vaatii paljon saman koodin toistamista, esim. useasti käytettävät värien heksa-

desimaaliarvot on kirjoitettava joka paikkaan uudestaan. Muut kehittäjät ovat havainneet samat ongelmat ja ratkaisseet ne toteuttamalla molemmille kielille esikäsittelijöitä. Niitä on olemassa myös JavaScriptille (esim. CoffeeScript), mutta ne eivät kuulu työn rajaukseen.

Esikäsittelijä on sovellus, joka muuntaa yhdentyyppisen datan toisenlaiseksi (Howe 2012). Dokumentit kirjoitetaan esikäsittelijöiden omilla kielillä ja käännetään sovelluksella tavalliseksi merkkaukseksi tai tyyleiksi. Ne on tarkoitettu nopeuttamaan front-end-koodin kirjoittamista ja vähentämään toiston tarvetta. Lisäksi esikäsittelijöiden käyttö parantaa koodin laatua, koska ne ovat hyvin tarkkoja syntaksivirheistä. Kääntäminen ei onnistu, jos esim. Jade-dokumentin sisennykset on merkattu väärin.

CSS-esikäsittelijät, joita ovat SASS, Stylus ja Less, tarjoavat useita CSS-kielestä puuttuvia ominaisuuksia. HAML, Jade ja muut HTML-esikäsittelijät yksinkertaistavat merkkauksen syntaksia huomattavasti. Pidän erityisesti siitä, ettei tageja tarvitse lopettaa tai ympäröidä erikoismerkeillä Jadessa, pelkkä sisentäminen riittää. Koodiesimerkissä 5 verrataan HTML- ja Jade-esikäsittelijän merkkausta.

HTML:

```
<!doctype html>
<html>
  <head><title>Hello world</title></head>
  <body>
    <div class="main" role="main">
```

JADE:

```
!!!
html
  head
    title Hello world
body
  .main(role='main')
```

KOODIESIMERKKI 5. HTML- ja Jade-merkkauksen vertailu

6 HTML

Alaluvuissa 6.1–6.4 käsitellään merkkauksen käytäntöjä. Näihin kuuluvat sivusto- ja sovelluksen rakentaminen laadukkaan pohjan päälle, toimeksiantajan hyödyntämän Jade-esikäsittelijän käyttö, yleinen merkkaustyyli (esim. elementtien sekä määritteiden käyttö ja dokumenttien organisointi) ja HTML:n 5-version hyödyntäminen. Vaikka kirjoitushetkellä HTML5-merkkauksen spesifikaatio ei ole valmis ja muuttuu jatkuvasti, voi uusia elementtejä ja määritteitä mielestäni käyttää, kunhan huomioi keskeneräisyyden ja pysyy ajan tasalla muutoksista.

6.1 Dokumenttipohja

HTML-dokumentit kannattaa rakentaa tietynlaisen mallipohjan päälle, jossa ei mielestäni saa olla liikaa merkkausta, vaan pelkästään tarvittavat tagit, elementit ja tekniikat, joita tarvitaan modernin, mukautuvan sivuston tai sovelluksen rakentamiseen. Jos asettelu tehdään aina saman rakenteen mukaan, silloin pohjaan voidaan toki sisällyttää enemmän merkkausta.

Twitter Bootstrap sisältää oman mallipohjansa, mutta vaihtoehtoja löytyy tässäkin tapauksessa. Itse suosin HTML5 Boilerplaten (Twitter Bootstrapia kevyempi front-end-sovelluskehys, http://html5boilerplate.com), pohjaa, joka on Bootstrapin pohjaa kattavampi. Alla esitellään tähän työhön liittyviä dokumenttipohjan merkkauksen tekniikoita.

Dokumenttityyppi (doctype)

HTML-dokumentin merkkaus alkaa dokumenttityypin määrittämisestä. Dokumenttityyppi kertoo selaimelle, millä HTML:n versiolla dokumentti on merkattu. Sen puute aiheuttaa ongelmia Internet Explorerilla mm. asettelun kanssa, koska selain asettaa sivuston tai sovelluksen standardeja noudattamattomaan Quirks-tilaan. Nykyään kaikki dokumenttipohjat, joita itse olen käyttänyt, sisältävät versioimattoman HTML5-tunnisteen. Koodiesimerkissä 6 on mallit XHTML 1.0- ja HTML5-dokumenttityypeistä.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!DOCTYPE html>
```

KOODIESIMERKKI 6. XHTML 1.0 - ja HTML5-doctype

Ehdolliset kommentit (conditional comment)

Joissain tilanteissa voi olla tarpeen määritellä CSS-tyylejä pelkästään joillekin Internet Explorerin versioille. Tämä voidaan tehdä esim. linkittämällä tyylitiedosto ehdollisten kommenttien sisällä. Kommenttien sisään kirjoitettu merkkaus on käytössä vain kommentissa määritellyillä IE:n versioilla. Mahdolliset operaattorit ovat:

- lt: pienempi kuin
- lte: pienempi tai yhtä suuri kuin
- gt: suurempi kuin
- gte: suurempi tai yhtä suuri kuin
- ei operaattoria: pelkästään tietty versio.

Koodiesimerkissä 7 kuvataan pelkästään Internet Explorerin 8-versiolla ladattavaa CSS-dokumenttia.

```
<!--[if IE 8]>
  <link rel="stylesheet" href="ie8.css">
<![endif]-->
```

KOODIESIMERKKI 7. Ehdollisesti ladattava CSS-dokumentti

Edellä mainittua tapaa ei näe dokumenttipohjissa, vaan sama ratkaisu on toteutettu kirjoittamalla HTML-tagista useita versiota kommenttien sisään. Mielestäni kyseistä tekniikkaa hyödyntäessä jokaista Internet Explorerissa esiintyvää ulkoasun virhettä ei tulisi korjata uudella tyylillä, vaan korjaukset kannattaa tehdä kaikkiin selaimiin vaikuttaviin tyyleihin. Samalla todennäköisesti oppii enemmän

selainyhteensopivan CSS:n kirjoittamisesta. Koodiesimerkissä 8 kuvataan edellä mainittu tekniikka ehdollisten kommenttien hyödyntämisestä.

```
// index.html
<!--[if lt IE 7]>      <html class="lt-ie9 lt-ie8 lt-ie7"> <![endif]-->
<!--[if IE 7]>         <html class="lt-ie9 lt-ie8"> <![endif]-->
<!--[if IE 8]>         <html class="lt-ie9"> <![endif]-->
<!--[if gt IE 8]><!--> <html> <!--<![endif]-->

// style.css
.lt-ie8 .header {
  /* IE8:aa vanhemmille versioille osoitetut tyylit */
}
```

KOODIESIMERKKI 8. HTML-tagin ympärille lisätyt ehdolliset kommentit

Meta-tagit

X-UA-Compatible-tagilla voidaan määrittää mitä selainmoottoria Internet Explorer käyttää oletuksena, jos vaihtoehtoja on useampi. Boilerplaten tagissa määritellään, että IE:n tulee käyttää uusinta (edge) versiota selainmoottorista. X-UA-Compatible-tagi kuvataan koodiesimerkissä 9.

```
<meta http-equiv="X-UA-Compatible" content="IE=edge">
```

KOODIESIMERKKI 9. Internet Explorerin käyttämän selainmoottorin määrittäminen

Sama voidaan määrittää myös back-end-koodissa tai palvelimen konfiguraatiossa. Havaitsin eräässä projektissa, ettei tagi toiminut käyttäessä sivustoa asiakkaan testiympäristössä, vaan oletukseksi asettui IE:n versio 7:n selainmoottori. Tämä johtui siitä, että IE:ssä on päällä asetus, joka näyttää intranetissä olevat sivustot yhteensopivuustilassa (Compatibility Mode). Koska sivusto ei sijainnut portissa 80, tulkitsi selain sen olevan intranetissä.

Mobiililaitteita varten hyödyllinen tagi on viewport. Niissä selainikkuna voi olla leveämpi kuin näytön leveys, jolloin käyttäjän täytyy vierittää sivua vaakasuunnassa nähdäkseen kaiken sisällön. Tämä ei ole käytettävyyden kannalta suotavaa mukautuvissa käyttöliittymissä. Width-parametrin device-width-arvolla selaimen leveys asetetaan laitteen leveydeksi. Initial-scale-parametri asettaa lähtötason selaimen skaalaukselle. Arvolla 1 varmistetaan, että kaikissa laitteissa sivua ei ole skaalattu oletuksena isommaksi tai pienemmäksi. Viewport-tagi ja edellä mainitut määritteet kuvataan koodiesimerkissä 10.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

KOODIESIMERKKI 10. Selainikkunan leveyden ja skaalauksen määrittäminen

6.2 Jade-esikäsittelijä

Toimeksiantaja käytti aiemmissa projekteissa sivustojen ja sovellusten sisältöpohjien merkkaukseen Java Server Pages -kieltä. Itselläni ei ole sen käytöstä paljoakaan kokemusta, koska en ole juurikaan ylläpitänyt tai tehnyt jatkokehitystä kyseisiin projekteihin. Kehittäjien mukaan Jadeen siirtymiseen oli muutamia hyviä syitä. Syntaksi on yksinkertaisempi, mikä johtaa selkeämpiin dokumentteihin. Jaden tuottamaan HTML:ään on helpompi vaikuttaa esim. tyhjän tilan osalta. Lisäksi siinä on sisäänrakennettuna enemmän ominaisuuksia kuin JSP:ssä, johon joudutaan usein hankkimaan erillisiä liitännäisiä. Koodiesimerkissä 11 on lyhyt esimerkki JSP:n syntaksista.

```
<ul>
<% for (int i = 1; i < menu.length; i++) { %>
  <a href="<%= menu.url %>"><%= menu.name %></a>
<% } %>
</ul>
```

KOODIESIMERKKI 11. JSP:n syntaksi

Alaluvussa 5.2 oli esimerkki Jaden syntaksista. Useimpien toimintojen käyttötavat riippuvat pitkälti projektista, jonka vuoksi esittelen alla pelkästään merkkauksen organisointiin ja ylläpidettävyyteen vaikuttavia. Täydellinen referenssi on luettavissa Jaden sivustolla (http://jade-lang.com/reference).

Sekoitukset (mixin)

Sekoitukset on tarkoitettu uudelleenkäytettävän merkkauksen luomiseen. Niihin voi sisällyttää parametrejä, toimintalogiikkaa ja Jade-merkkausta. Sekoituksen kutsuun lisättyä luokka- tai id-attribuuttia voi käyttää ilman, että niitä on erikseen määritelty parametreiksi. Sekoitusten käyttöä kuvataan koodiesimerkissä 12.

Jade:

```
mixin input(type, name)
  input(class=attributes.class, type=type, name=name)

+input('text', 'login').input-text
```

HTML:

```
<input class="input-text" name="login" type="text">
```

KOODIESIMERKKI 12. Jade-sekoitus

Lohkot (block)

Lohkot ovat Jade-dokumenttiin määriteltäviä osia, joiden alkuun (prepend) tai loppuun (append) voi lisätä merkkausta ilman osan ylikirjoittamista. Ne ovat erityisen hyödyllisiä sellaisen sisällön kohdalla, josta osa on dokumenttipohjassa ja osa määritellään dokumenttikohtaisesti. Esimerkkinä sivuston header-osio, jonka otsikko on sivukohtainen. Lohkojen käyttöä kuvataan koodiesimerkissä 13.

Jade:

```
// base.jade
header
  block pagetitle
    img(src='logo.jpg')

// index.jade
extends base
block append title
  h1 Hello world!
```

HTML:

```
<header>
  <img src="logo.jpg">
  <h1>Hello world!</h1>
</header>
```

KOODIESIMERKKI 13. Jade-lohko

Sisällyttäminen (include)

Kokonaisia Jade-dokumentteja voidaan sisällyttää toisiin dokumentteihin sisällytystoiminnolla. Huonona puolena on se, ettei dokumentista voi valita tiettyjä osia sisällytettäväksi. Se soveltuu tästä huolimatta mielestäni hyvin dokumenttien organisointiin, koska merkkaus voidaan jakaa useampaan osaan. Sisällyttämistä kuvataan koodiesimerkissä 14.

```
// scripts.jade
script(src="jquery.js")
script(src="main.js")

// base.jade
include scripts
```

KOODIESIMERKKI 14. Sisällyttäminen Jadessa

6.3 Merkkaustyyli

Sivustot ja sovellukset vaativat usein suuren rivimäärän vaihtelevan sisällön merkkausta. Kuten muunkin front-end-koodin tapauksessa, useat kehittäjät saattavat muokata samoja dokumentteja. Ylläpidettävän ja luettavan merkkauksen saavuttamiseksi tulee huomioida elementtien oikeellista käyttöä, koodin erottelemista, esteettömyyttä, määritteiden (attribute) järjestystä ja dokumenttien organisointia.

Elementtien käyttö

Front-end-kehityksen ulkopuolella semantiikka tarkoittaa merkkien ja niiden tarkoitteiden suhteiden tutkimusta. Kielitieteessä termi on määritelty sanojen, lauseiden ym. kielen ilmausten merkityksen tutkimukseksi (SuomiSanakirja 2013). Merkkauksen semantiikasta puhuttaessa tarkoitetaan elementtien, määritteiden ja määritteiden arvojen sovittuja merkityksiä. Vaikka merkitykset virallistetaan usein W3C:n toimesta HTML:n spesifikaatioon, saattavat ne kehittyä ja muuttua ajan myötä kehittäjien käsissä. Semantiikka auttaa ensisijaisesti ohjelmia, kuten hakukoneita ja näytönlukijoita ymmärtämään sivustojen ja sovellusten sisältöä, mutta myös ihmiset hyötyvät siitä (Gallagher 2012a). Gallagher (2012a) pitää semanttisen merkkauksen kirjoittamista yhtenä nykyaikaisen ja ammattimaisen front-end-kehityksen perusteista.

Tarkoituksena on pitää merkkaus kuvauksena sisällöstä, eikä siitä, miltä sisällön tulisi näyttää. Tähän kuuluu näkemykseni mukaan lisäksi tietynlainen minimalismi, jossa sisältö pyritään esittämään mahdollisimman pienellä määrällä merkkausta, esim. karsimalla merkityksettömiä div-elementtejä. Elementtien merkityksen ymmärtämisessä alkuun pääsee tutustumalla HTML5:n spesifikaatioon (http://www.w3.org/TR/html5/dom.html#dom) tai mielestäni selkeämpiin oppaisiin, kuten Shay Howen A Beginner’s Guide to HTML & CSS (http://learn.shayhowe.com/html-css/elements-semantics). Koodiesimerkissä 15 havainnollistetaan semantiikan noudattamisen tärkeyttä.

```
<font size="10"><b>Page title</b></font>
```

→

```
<h1>Page title</h1>
```

KOODIESIMERKKI 15. Kaksi eri tapaa sivun otsikon merkkaukseen

Mielestäni jo lyhyt esimerkki tuo hyödyt esille. Koodista tulee lyhempää ja ulkoasun määrittely pysyy sille tarkoitetussa paikassa, CSS-tyyleissä. Visuaalisuuteen liittyvät määritteet on poistettu kokonaan HTML5-spesifikaatiosta. Myös jotkin elementit on poistettu tai niille on määritelty uusi merkitys (W3C 2013a).

Näen semanttisen merkkauksen olevan osa esteettömyyttä, mobile-first-ajattelutapaa ja pohja vaiheittaiselle parantamiselle. Järkevästi merkattu ja merkityksellinen sisältö on kaikkien käyttäjien saatavilla ja ymmärrettävää päätelaitteen ominaisuuksista ja fyysisistä rajoitteista huolimatta. Hyvä testi on ottaa selaimen asetuksista CSS-tyylien tuki pois käytöstä ja tutkia, onko sivuston sisältö luettavaa.

Kuten muidenkin käytäntöjen noudattamisessa, semantiikan tavoittelussa ei kannata mennä liiallisuuksiin. Jos ylimääräistä merkkausta karsii liikaa, ulkoasun määrittely tai JavaScript-toimintojen toteuttaminen saattaa osoittautua hankalaksi.

Organisointi

Jadea tai muuta sisältöpohjakieltä käyttämällä dokumentit on helppo pitää hyvin selkeinä ja jaoteltuina. Osiot, kuten header ja footer, voidaan jakaa omiin dokumentteihinsa ja sisällyttää dokumenttipohjaan. Yleisesti käytetyt komponentit, jotka vaativat useamman rivin merkkausta, kannattaa määritellä sekoituksiin. Verkkokauppasovelluksessamme näitä ovat esim. yhteystietolomake ja ostoskorin rivit. Vaikka sisällytystä voi hyödyntää samaan tarkoitukseen, se ei kaikissa tapauksissa ole kannattavaa. Jokaista komponenttia varten joutuu luomaan

oman dokumenttinsa, koska pelkästään kokonaisia dokumentteja voi sisällyttää, ei yksittäisiä osia.

Koodin erottelu

Merkkaukseen on mahdollista sisällyttää CSS-tyylejä ja JavaScript-merkkausta. En pidä sitä kuitenkaan suositeltavana käytäntönä, koska sisällytys sekoittaa koodin organisointia ja vaikeuttaa ylläpitoa. Luokka- id-, ja HTML5:n data-määritteet tarjoavat riittävän rajapinnan JavaScript-koodissa elementtien manipuloimiseen ja tapahtumiin, kuten linkkien painamiseen. CSS:n sisällytys tapahtuu joko style-tagilla tai -määritteellä. CSS:n sisällytystä kuvataan koodiesimerkissä 16.

```
<style type="text/css">
  .class {
    font-size: 13px;
  }
</style>

<div style="width: 100%; margin: 10px 0;">
  …
</div>
```

KOODIESIMERKKI 16. Style-tagi ja -määrite

JavaScriptiä voidaan sisällyttää HTML:n event-määritteillä (esim. onclick). Toinen tapa on koodin kirjoittaminen script-tagien sisään. JavaScriptin sisällytystä kuvataan koodiesimerkissä 17.

```
<a href="#" onclick="foo(bar);">Osta tuote</a>
<script>
  function foo (bar) {
    …
  }
</script>
```

KOODIESIMERKKI 17. Event-määrite ja script-tagi

Kuten muissakin käytännöissä, poikkeustilanteitakin löytyy. Eräässä projektissa jouduin kirjoittamaan back-end-puolelta tuotavia muuttujia, esim. lokalisoitavia tekstejä script-tagiin, koska en löytänyt tapaa käyttää niitä suoraan JavaScript-dokumenteissa. Myös useiden JavaScript-sisältöpohjien kanssa pohjat on sisällytettävä merkkaukseen. Molemmissa tapauksissa koodi voidaan silti pitää selkeyden vuoksi omissa dokumenteissaan.

Määritteiden järjestäminen

HTML:n lukuisia määritteitä varten kannattaa sopia käytäntö niiden kirjoitusjärjestyksestä. Monimutkaisen säännön sijasta mielestäni riittää, että yleisimmin käytetyt määritteet ovat joka kerta samassa järjestyksessä. Tämä nopeuttaa kokemukseni mukaan kirjoitusnopeutta sekä luettavuutta ja yhdenmukaistaa osaltaan dokumentteja. Oppaassa järjestys on määritelty seuraavasti:

1. luokka
2. id
3. data-määritteet
4. muut määritteet.

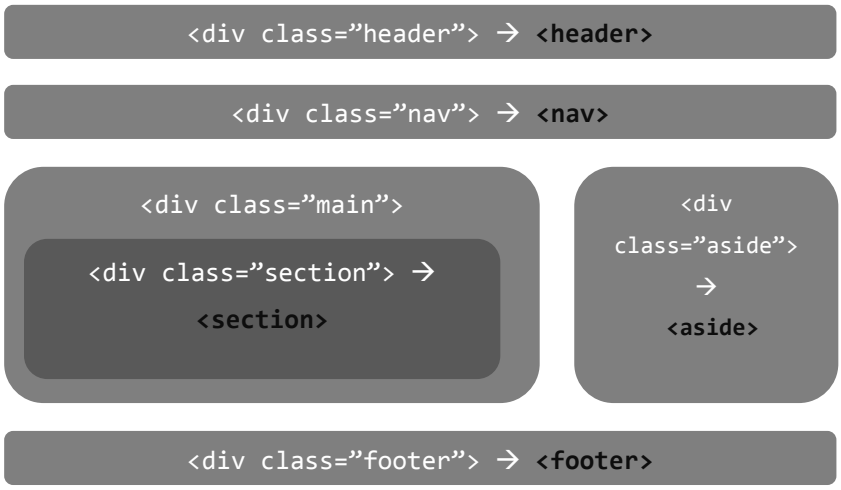
6.4 HTML5:n hyödyntäminen

Toimeksiantaja käyttää HTML5-spesifikaation monia uusia elementtejä ja määritteitä jo nykyisissä projekteissa ja varmasti myös jatkossakin. Niiden käytöllä voidaan vaikuttaa mm. semantiikkaan, käytettävyyteen sekä esteettömyyteen ja helpottaa JavaScript-toimintojen toteuttamista. Tämän tasoisessa käytössä on otettava muutama seikka huomioon selaintuen kanssa, koska vanhemmat selaimet eivät tunnista uusia elementtejä.

Elementit

Selain osaa piirtää sille tuntemattomat merkkauksessa olevat elementit, mutta ei osaa tulkita niille määritettyjä CSS-tyylejä. Ongelma on ratkaistavissa käyttämällä valmista JavaScript-kirjastoa (esim. HTML5 shiv, http://remysharp.com/2009/01/07/html5-enabling-script), joka luo uudet elementit DOM-puuhun. Lisäksi HTML5:n lohkotason elementeille täytyy erikseen määrittää `display: block` -deklaraatio, etteivät ne jää vanhemmissa selaimissa tekstitason (inline) elementeiksi. Molemmat ratkaisut ovat mukana esim. HTML5 Boilerplate -sovelluskehyksessä.

Ulkoasun asetteluun liittyvillä elementeillä voidaan korvata osa div-elementeistä. Niistä on eniten hyötyä hakukoneille, näytönlukijoille ja muille sivua lukeville ohjelmille, mutta mielestäni ne myös selkeyttävät koodia kehittäjän näkökulmasta. Kuvassa 5 on esimerkki siitä, kuinka div-elementtejä voidaan korvata HTML5:n elementeillä sivun asettelussa.



KUVA 5. Asettelun toteuttaminen HTML5-elementeillä

Loput elementit, jotka ovat täysin uusia ja joiden merkitystä on muutettu, ovat dokumentoituna HTML5 Doctor -sivustolla (http://html5doctor.com/element-index)

Määritteet

Toimeksiantajan käyttö rajoittuu uusien määritteiden osalta lomakkeiden kenttien rikastamiseen ja mahdollisuuteen tehdä omia määritteitä. HTML5:n tarjoamat lomakkeen kenttien määritteet ovat hyödyllisiä erityisesti mobiilikäytössä, jossa ne tekevät lomakkeista käytettävämpiä. Lyhyenä esimerkkinä ovat uudet kenttätyypit, jotka on myös kuvattu HTML5 Doctor -sivustolla (http://html5doctor.com/html5-forms-input-types). Holstin (2013) mukaan huono tai epäjohdonmukainen lomakkeiden toteutus vaikuttaa mobiilikäytössä suuresti käytettävyyteen ja jopa sivuston tai sovelluksen luotettavuuteen käyttäjien silmissä.

Mobiililaitteet näyttävät kentän tyypin mukaan määräytyvän näppäimistön. Vaikka näppäimistöä voi vaihtaa, säästää sopivan näyttäminen oletuksena aikaa varsinkin, jos lomakkeella on runsaasti kenttiä. Lisäksi joissain näppäimistössä painikkeiden koot ovat suurempia, mikä auttaa vähentämään virhepainallusten määrää. Jotta käyttäjä ei hämäänny turhaan, kenttätyypit tulee määritellä johdonmukaisesti, eli puhelinnumerokenttien tulee olla aina tel-tyyppisiä ja sähköpostikenttien email-tyyppisiä jne. Selaintuen kanssa ei aiheudu ongelmia, koska vanhemmat selaimet tulkitsevat HTML5:n kentät tekstityyppisinä. Koodiesimerkissä 18 kuvataan, kuinka tekstikenttä muutetaan puhelinnumerokentäksi.

```
<label for="phone">Puhelinnumero</label>
<input type="text" name="phone">
```

→

```
<label for="phone">Puhelinnumero</label>
<input type="tel" name="phone">
```

KOODIESIMERKKI 18. Puhelinnumero-kentän määrittäminen

Mahdollisuus omien määritteiden tekemiseen on käytettävyyden parantamisen sijasta hyödyllisempää JavaScript-koodauksessa. Merkkauksesta ei tarvitse tarjoilla dataa JavaScriptille pelkästään luokka-, id- tai muissa määritteissä, vaan siihen voi käyttää räätälöitäviä data-määritteitä. Data-määritteen nimen on oltava vähintään yhden merkin pituinen ja siinä on oltava etuliite "data-". Nimes-

sä ei saa olla isoja kirjaimia. Määritteen arvo voi olla millainen tahansa merkkijono. Selain ei käsittele data-määritteitä millään tavalla (Bewick 2010). Koodiesimerkissä 19 kuvataan data-määritteen käyttämistä merkkauksessa ja JavaScriptissä jQuery-kirjaston avulla.

HTML:

```
<button class="btn btn-add-to-cart" data-product-id="1012">Lisää koriin
</button>
```

JavaScript + jQuery-kirjasto:

```
var productId = $('.btn-add-to-cart').data('product-id');
```

KOODIESIMERKKI 19. Data-määritteen käyttäminen

HTML5-spesifikaation mukaan data-määritteet on tarkoitettu pelkästään sivustolle tai sovellukselle kuuluvan datan säilytykseen, jolle ei ole olemassa sopivampia määritteitä tai elementtejä. Jokaisella elementillä voi olla kuinka monta data-määritettä tahansa (W3C 2013b). Näkemykseni mukaan data-määritteissä olisi hyvä säilyttää pelkästään käyttäjälle turhaa dataa, jota ei haluta esittää käyttöliittymässä sellaisenaan. Verkkokauppaprojektissa niitä käytetään usein back-end-puolen datan, kuten esimerkin mukaisen tuotteen tunnisteen tarjoamiseen JavaScript-koodille.

7 CSS

CSS-dokumenttien koko saattaa kasvaa nopeasti satoja rivejä pitkäksi. Näkemykseni mukaan rivimäärän kasvaminen johtuu osittain CSS:n puutteista ja kielen tietynlaisesta alkeellisuudesta. Esimerkiksi kaikki arvot on toistettava joka kerta uudestaan. Se ei sisällä ohjelmointikielistä tuttuja ominaisuuksia, kuten muuttujia, funktioita tai laskuoperaatioita. Korpelan ja Lehtovirran (2013, 98) mukaan kyse ei ole pelkästään koodaamiseen helppoudesta, vaan koodin ylläpidettävyydestä, kehitettävyydestä ja kopioinnista johtuvien virheiden välttämisestä. Puutteita voidaan paikata käyttämällä CSS-esikäsittelijöitä.

7.1 Tekniikat

Laatikkomalli

Laatikkomallin mukaan jokainen sivulla oleva HTML-elementti on suorakuutio, jolla voi olla marginaalit (margin), täytettä (padding) ja reunukset (border) (Howe 2012). Koodiesimerkissä 20 kuvataan laatikkomalliin liittyvät CSS-ominaisuudet.

CSS:

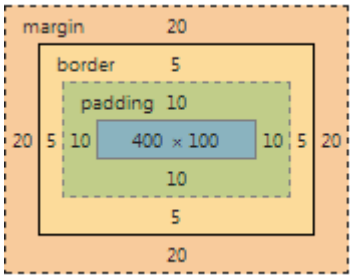
```
.a {
  width: 400px;
  height: 100px;
  margin: 20px;
  padding: 10px;
  border: 5px solid #ddd;
}

.b {
  width: 100%;
  padding: 20px;
  …
}
```

HTML:

```
<div class="a">
  <div class="b"></div>
</div>
```

KOODIESIMERKKI 20. Laatikkomallin ominaisuudet ja sisäkkäiset elementit

Modernit selaimet tulkitsevat laatikkomallin oletuksena niin, että elementin koko leveydeksi ja korkeudeksi tulee width- ja height-arvojen, täytteen ja reunojen yhteenlaskun tulos. Eli esimerkissä elementin leveys on 400 + 40 + 10 = 450, ei 400 pikseliä. Marginaaleja ei lasketa mukaan. Kuvassa 6 havainnollistetaan elementin laatikkomallia.



KUVA 6. A-elementin laatikkomalli Google Chromen kehitystyökalujen esittämänä

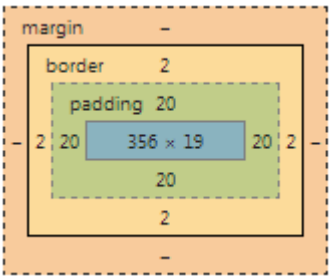
Kyseinen tulkinta voi aiheuttaa päänvaivaa varsinkin mukautuvissa käyttöliittymissä. Esimerkkinä tapaus, jossa mobiililaitteilla sisältöalueen ja lomakkeen kenttien leveydeksi on määritelty 100 % selaimen leveydestä. Jos lomakkeille määrittää täytettä, ne voivat venyä yli sisältöalueesta kuvan 7 tapaan.

KUVA 7. Sisäkkäiset lohkoelementit

Ongelma on ratkaistavissa muuttamalla selaimen laatikkomallin tulkintaa CSS-määritteellä, jolloin elementin koko ulottuvuus määräytyy width- ja height-arvojen mukaan, eikä yhteenlaskun tuloksesta. Itse pidän kyseistä tapaa loogisempana ja helpommin ymmärrettävänä. Twitter Bootstrap -sovelluskehyksessä kaikille elementeille on annettu kyseinen deklaraatio yhdenmukaisuuden vuoksi. Koodiesimerkissä 21 on malli deklaraatiosta. Kuvassa 8 havainnollistetaan, kuinka deklaraatio vaikuttaa elementin laatikkomalliin.

```
.b {
  box-sizing: border-box;
  …
}
```

KOODIESIMERKKI 21. Laatikkomallin määrittäminen



KUVA 8. B-elementin laatikkomalli, leveys yhteensä 400 pikseliä

Selainkohtaisten tyylierojen normalisointi

Selaimet sisältävät oman tyylitiedostonsa (user agent style sheet). Niiden tyylit on kuvattu esim. IECSS-sivustolla (http://www.iecss.com). Tyyleillä määritetään oletusulkoasu elementeille (mm. linkkien sininen väri) ja varmistetaan, että selaimet pystyvät näyttämään sisältöä vaikka sivusto tai sovellus ei sisältäisi yhtään riviä CSS-koodia. Oletustyylit täytyy aina ajaa erikseen yli omilla tyyleillä, ja niissä on paljon selainkohtaisia eroja, joka tekee ylikirjoittamisesta työlästä.

Manuaalisen työn sijasta kannattaa hyödyntää valmista CSS-dokumenttia, joiden säännöillä vaikutetaan selainkohtaisiin eroihin. Vanhempi tapa on Eric Meyerin (2011) kehittämä nollaaminen, CSS Reset, joka poistaa kaikkien elementtien oletustyylit. Koodiesimerkissä 22 on lyhyt esimerkki nollaamisesta.

```
html, body, div …
h1, h2 … {
  margin: 0;
  padding: 0;
  border: 0;
  font-size: 100%;
  font: inherit;
  vertical-align: baseline;
}
```

KOODIESIMERKKI 22. CSS Reset -tyylitiedoston koodia

Uudempi versio nollaamisesta on tyylien normalisointi (normalize). Gallagher (2012b) kuvaa tekniikan tavoitteita: hyödyllisten oletustyylien säilyttäminen kaikkien poistamisen sijaan, bugien ja epäjohdonmukaisuuksien korjaaminen elementtien ulkoasussa, käytettävyyden parantaminen ja koodin selostaminen kommenteilla sekä dokumentaatiolla. Normalisointi eroaa nollauksesta edellä mainittujen kohtien osalta, joka tekee siitä mielestäni paremman vaihtoehdon. Normalize-dokumentti on usein valmiina osana front-end-sovelluskehyksiä.

7.2 CSS-esikäsittelijät

Luvun alussa kuvattujen puutteiden takia on hyödyllistä käyttää apuna CSS-esikäsittelijää. Kirjoitushetkellä vaihtoehtoja on kolme: Sass, LESS ja Stylus. Koodi kirjoitetaan tyylitiedostoihin, jotka kirjoitetaan esikäsittelijän syntaksilla ja käännetään tavalliseksi CSS-kieleksi. Kuvaan keskeisimpiä ominaisuuksia ja esikääntäjien käyttöä tarkemmin seuraavissa luvuissa.

7.2.1 Ominaisuudet

Vaikka itse olen käyttänyt pelkästään LESS-esikäsittelijää, tarjoavat eri vaihtoehdot näkemykseni mukaan nykyään pitkälti samat ominaisuudet, ja suurimmat erot ovat enää syntaksissa. Pidän kaikkia olennaisesti yhtä hyvinä vaihtoehtoja. Valinta kannattaa mielestäni tehdä toimintojen eroavaisuuksien sijasta syntaksin perusteella. Toinen kriteeri on käytetyn front-end-sovelluskehyksen esikäsittelijä. Toimeksiantaja käyttää LESS:iä, koska sitä käytetään Twitter Bootstrapissa.

Alla esitellään muutamia keskeisimpiä ominaisuuksia. Loput ovat löydettävissä kielien dokumentaatiosta. Esimerkit on kirjoitettu LESS:n syntaksilla.

Less:	http://lesscss.org
SASS:	http://sass-lang.com
Stylus:	http://learnboost.github.io/stylus

Muuttujat

Muuttujat ovat kutsuttavissa kaikkialla dokumentissa, jossa ne on asetettu. Niille voi antaa arvoksi minkä tahansa CSS-arvon (esim. värin, numeron yksikköineen tai tekstin), toisen muuttujan tai laskutoimituksen tuloksen. Muuttujia kannattaa mielestäni käyttää runsaasti ulkoasua määrittävissä tyyleissä, kuten väreissä ja fonteissa, joka nopeuttaa vaihtelevien teemojen toteuttamista sivustoille ja sovelluksille. LESS-muuttujien käyttöä kuvataan koodiesimerkissä 23.

LESS:

```
@base-font-size: 14px;
@large-font-size: @base-font-size + 2;
@font-color: #222;

body {
  font-size: @base-font-size;
  color: @font-color;
}
```

```
h1 {
  font-size: @large-font-size;
}
```

CSS:

```
body {
  font-size: 14px;
  color: #222;
}

h1 {
  font-size: 16px;
}
```

KOODIESIMERKKI 23. LESS-muuttujat

Sisäkkäisyys (nesting)

HTML-elementtiin ja sen lapsi- tai viereisiin elementteihin kohdistuvat selektorit voidaan kirjoittaa sisäkkäin. Sisäkkäisyyttä kuvataan koodiesimerkissä 24.

LESS:

```
.header {
  h1 { … }
  p {
    a {
      &:hover { … }
      …
    }
    …
  }
}
```

CSS:

```
.header { … }
.header h1 { … }
.header p { … }
.header p a { … }
.header p a:hover { … }
```

KOODIESIMERKKI 24. LESS:n sisäkkäisyys

Esimerkki toimii myös mallina vältettävästä tavasta, jossa luettavuus kärsii liian monen sisennystason takia. Sisennys tulee mielestäni pitää korkeintaan kaksi-kolme tasoa syvänä.

Sekoitukset

Sekoitukset ovat funktioita, jotka mahdollistavat deklaraatioiden uudelleenkäytön säännöissä. Hyvä käytäntö on koota toisiinsa liittyvät sekoitukset Twitter Bootstrapin tapaan omiin dokumentteihinsa. Sekoituksia kuvataan koodiesimerkissä 25.

LESS:

```
.border-radius(@radius) {
  -moz-border-radius: @radius;
  border-radius: @radius;
}

.btn {
  .border-radius(4px);
}
```

CSS:

```
.btn {
  -moz-border-radius: 4px;
  border-radius: 4px;
}
```

KOODIESIMERKKI 25. LESS-sekoitukset

Periytyvyys

Sääntöihin voidaan muuttujien ja sekoitusten lisäksi periyttää toisia sääntöjä. LESS-kielessä periytetyt säännöt toimivat kuin sekoitukset ilman parametrejä. Periytyvyyttä kuvataan koodiesimerkissä 26.

LESS:

```
.centered {
  display: block;
  margin: 0 auto;
}

h1 {
  .centered;
}
```

CSS:

```
h1 {
  display: block;
  margin: 0 auto;
}
```

KOODIESIMERKKI 26. LESS:n periytyvyys

Pidän kyseistä ominaisuutta sääntöjen organisoinnin lisäksi kätevänä työkaluna merkkauksen siistimiseen. Merkkaukseen ei välttämättä tarvitse lisätä elementeille useita luokkia, vaan ne voi periyttää säännöissä, joka nähdään koodiesimerkissä 27.

HTML:

```
<button class="btn btn-primary btn-large btn-save">Tallenna</button>
```

→

```
<button class="btn-save">Tallenna</button>
```

LESS:

```
.btn-save {
  .btn;
  .btn-primary;
  .btn-large;
}
```

KOODIESIMERKKI 27. CSS-sääntöjen organisointi

Tuonti

Käytän harvoin CSS:n omaa tuontiominaisuutta, koska se vaatii http-pyynnön jokaiselle tuodulle dokumentille. Esikääntäjien tuonti tapahtuu eri tavalla: koko dokumentin sisältö sisällytetään toiseen käännösvaiheessa, jonka tuloksena on yksi dokumentti. Siinä on käytettävissä tuoduissa dokumenteissa asetetut muuttujat ja sekoitukset. Muuttamalla kolmannen osapuolen tyylitiedostot LESS-dokumenteiksi (ei vaadi muuta kuin tiedostopäätteen muuttamisen), nekin voidaan yhdistää suorituskyvyn parantamiseksi ilman erillisiä back-end-puolen tekniikoita. Tuontiominaisuutta kuvataan koodiesimerkissä 28.

LESS:

```
// tables.less
table {
  border-collapse: collapse;
}

// style.less
@import 'tables.less';
```

CSS:

```
/* style.css */
table {
  border-collapse: collapse;
}
```

KOODIESIMERKKI 28. LESS:n tuonti

7.2.2 Kääntäminen

Dokumentit voidaan kääntää tavalliseksi CSS-kieleksi eri tavoilla: komentoriviltä, työpöytäsovelluksella (Windowsille esim. WinLess tai SimpLess, Mac OSX:lle CodeKit), WWW-sovelluksella (esim. http://winless.org/online-less-compiler) tai LESS:in tapauksessa front-end-puolella linkittämällä käännöksen tekevä JavaScript-kirjasto HTML-dokumenttiin. Jos käännös tehdään selaimessa, linkitetään merkkauksessa LESS-dokumenttiin CSS-dokumentin sijasta koodiesimerkin 29 mukaisesti.

```
<link rel="stylesheet/less" type="text/css" href="style.less">
```

KOODIESIMERKKI 29. LESS-dokumentin linkittäminen merkkauksessa

Komentoriviltä käännettäessä esikääntäjä on ensin asennettava paikallisesti tai palvelimelle. Kehitysvaiheessa on nopeampaa ja kätevämpää käyttää paikallista asennusta, jolloin jokaisen muutoksen jälkeen dokumentteja ei tarvitse siirtää versionhallintaan tai palvelimelle. Koodiesimerkki 30 sisältää mallin käännöskomennosta ja kahdesta mahdollisesta parametristä.

```
lessc input.less output.css [--compress, --watch]
```

KOODIESIMERKKI 30. LESS:n kääntäminen komentoriviltä

Compress-parametri pakkaa tuotetun output.css-dokumentin, ja watch jättää komentorivin auki tekemään käännöksen uudestaan aina, kun dokumenttiin tehdään muutoksia. Kääntäjä tunnistaa muutokset myös tuoduissa dokumenteissa. Käyttämäni työpöytäsovellukset toimivat samalla tavalla. Mahdollisia parametrejä on näiden kahden lisäksi monia muita, jotka on dokumentoitu LESS:n tutoriaalissa.

Tuotantovaiheessa voidaan käyttää kolmannen osapuolen sovelluksia tai ohjelmointikielien liitännäisiä palvelimella, jotka tekevät käännöksen ja muut toimenpiteet, kuten pakkauksen ja yhteenliittämisen automaattisesti sivuston tai sovelluksen uuden version kääntämisen yhteydessä. Itse kuitenkin suosin yk-

sinkertaisuuden vuoksi sitä, ettei versionhallinnassa säilytettäisi LESS-dokumentteja, vaan pelkkä käsitelty CSS-dokumentti.

7.3 Tyylien kirjoittaminen

Organisointi

Yksi hyvä malli tyylitiedostojen organisointiin löytyy Twitter Bootstrapista. Toisiinsa liittyvien elementtien koodi, muuttujat ja sekoitukset jaotellaan omiin dokumentteihinsa ja liitetään yhteen esikäsittelijän tuontitoiminnolla. Suosittelen määrittelemään tuodut dokumentit mahdollisimman lähellä dokumentin alkua, jolloin on helpompi välttää sääntöjen ylikirjoittumiseen liittyviä ongelmia. Olen huomannut, että ongelmia saattaa esiintyä varsinkin Bootstrapin komponenttien tyylejä räätälöidessä, jos valmiit tyylit ajavatkin itse kirjoitettujen yli. Koodiesimerkissä 31 on esimerkki tyylitiedostojen jaottelusta.

```
// typography.less
body {
  font-size: 14px;
}

// buttons.less
.btn {
  margin-bottom: 20px;
}
```

KOODIESIMERKKI 31. Tyylitiedostojen jaottelu

Alkuvaiheessa saattaa olla hankalaa ennustaa tulevaa koodin määrää, jolloin mielestäni riittää se, että säännöt kootaan yhteen dokumenttiin selkeillä kommenteilla merkattuihin osioihin. Sisällysluettelon tekeminen on hyödyllistä, jos osioita alkaa syntyä useita. Sen ylläpitäminen käsin saattaa käydä työlääksi, joten työn automatisointi voi olla järkevää, jos se on mahdollista käytetyssä ke-

hitystyökalussa tai tekstieditorissa. Koodiesimerkissä 32 on esimerkki tyylitiedoston organisoinnista sisällysluettelon ja otsikkojen avulla.

```
// style.less
/* --------------
TABLE OF CONTENTS
  1. Typography
  2. Components
---------------*/

// 1. TYPOGRAPHY
// -------------
…

// 2. COMPONENTS
// -------------
// Buttons
…
// Forms
…
```

KOODIESIMERKKI 32. Yksittäisen tyylitiedoston organisointi

Luokkien ja id-määritteiden nimeäminen

HTML:n luokka- ja id-määritteet toimivat kytkentöinä CSS-sääntöihin ja JavaScript-koodiin. Ne tuovat elementeille merkitystä ja toimivat erilaisten käyttöliittymäkomponenttien niminä, jonka vuoksi Gallagher (2012a) pitää niitä osana alaluvussa 6.3. kuvattua semantiikkaa, tässä tapauksessa enemmän kehittäjille kuin hakukoneille tarkoitettuna. HTML5-spesifikaatio määrittelee nimeämiseen liittyvän parhaan käytännön: nimien tulisi kuvata sisällön laatua ja luonnetta, ei sen ulkoasua (W3C 2012c). Gallagher (2012a) ei kuitenkaan näe tämän noudattamista pakollisena, vaan kokee kyseisen käytännön olevan usein hidaste työskennellessä laajojen sivustojen ja sovellusten parissa. Nimet ovat hyödyllisempiä ja antavat enemmän tietoa muille kehittäjille, jos niillä kuvaa muutakin kuin elementin sisältöä. Ulkoasua kuvaavilla luokilla saadaan aikaiseksi uudel-

leen käytettäviä ja muokattavia komponentteja, joita voidaan käyttää eri tarkoituksiin merkkauksessa.

Twitter Bootstrap käyttää ns. moniluokkamallia (multi-class pattern) nimeämisessä. Tarkoituksena on erotella käyttöliittymäkomponenttien pohja- sekä ulkoasuun liittyvät tyylit omiksi säännöikseen. Luokat voidaan määrittää joko suoraan elementille, tai yhdistää ne CSS-esikäsittelijän toiminnoilla (esim. LESS:n periytyvyys, esimerkki alaluvussa 7.2.1). Moniluokkamallia kuvataan koodiesimerkissä 33.

CSS:

```
.btn {
  font-size: 14px;
  padding: 5px;
  /* Muut kaikkiin painikkeisiin liittyvät tyylit */
}

.btn-primary {
  color: #B4D455;
  /* Muut pääpainikkeeseen liittyvät tyylit */
}
```

HTML:

```
<input class=”btn btn-primary” type=”submit” value=”Tallenna”>
<button class=”btn” type=”button”>Peruuta</button>
```

KOODIESIMERKKI 33. Twitter Bootstrapin nimeämismalli

Ominaisuuksien järjestys

Varsinainen sääntöjen sisältö, ominaisuudet, voidaan järjestää esimerkiksi aakkosjärjestykseen tai käyttötarkoituksen mukaan. Pidän käyttötarkoituksen mukaan järjestämistä parempana vaihtoehtona. Ideana on ryhmitellä ominaisuudet sen mukaan, mihin ne vaikuttavat (esim. asettelu, typografia, laatikkomalli). Tapa saattaa olla aluksi hankalampi muistaa, mutta lisää mielestäni selkeyttä ja tekee säännöistä paremmin rakennettuja ja luettavampia. Uusia ominaisuuksia

on nopeampaa lisätä vanhojen joukkoon kuin aakkosjärjestyksessä. Esikääntäjän sisäkkäiset säännöt ja sekoitukset kannattaa lisätä säännön alkuun samasta syystä kuin tuotuja dokumentteja määritettäessä. Tällöin lisättyjä tyylejä on helpompi ylikirjoittaa.

8 JAVASCRIPT

Ensimmäiset kokemukseni JavaScriptistä olivat jonkin verran negatiivisia, enkä pitänyt sitä oikeana ohjelmointikielenä, vaan huonosti toimivana ja hieman turhana, lähinnä popup-ikkunoita ja erilaisia laskureita luovana kielenä. Vasta työelämässä asiakasprojektien parissa olen tutustunut siihen kunnolla ja opetellut kieltä. Näkemykseni mukaan kielen, kirjastojen sekä työkalujen kehitys ja parempi selaintuki ovat tehneet siitä huomattavasti luotettavamman ja front-end-kehitykseen paremmin soveltuvan työkalun. Toimeksiantaja toteuttaa projekteissa nykyään yhä useammin aiemmin täysin back-end-puolella hoidettuja toimintoja (esim. verkkokaupassa tuotteiden listaus sekä haku ja ostoskorin muokkaus) front-end-puolelle JavaScriptillä. Back-end-puolen tehtäväksi jää näissä tapauksissa pelkästään datan tuominen taustajärjestelmistä tai tietokannasta käyttöliittymälle.

Olen huomannut, että JavaScript-koodin kirjoittaminen jää projekteissa usein toissijaiseen asemaan. Koodin laatuun ei kiinnitetä tarpeeksi huomiota, eikä käytäntöjä ole vielä määritelty ollenkaan. Tästä syystä otin myös asiakseni ottaa selvää hyvästä koodaustyylistä ja koodin laadun parantamisesta testaamisella.

8.1 Tekniikat

Kaksi tärkeää apuvälinettä, joita toimeksiantaja käyttää JavaScript-toimintojen toteuttamisessa, ovat jQuery (http://jquery.com) ja erilaiset MVC-sovelluskehykset. JQuery on JavaScriptiä laajentava sovelluskehys, joka sisältää laajan valikoiman toimintoja mm. HTML-elementtien valitsemiseen sekä muokkaamiseen, tapahtumien käsittelyyn, animaatioiden tekemiseen ja AJAX:n käyttöön. Pidän jQueryn vahvuutena erityisesti sen yksinkertaisuutta verrattuna perus-JavaScriptiin. Toimintoja on helpompi käyttää kuin JavaScriptin vastaavia, koska toimintalogiikkaa on piilotettu kirjaston lähdekoodiin. Suosittelen kuitenkin ennen kirjaston käyttämistä opiskelemaan JavaScriptin toimintaa ja ominaisuuksia, vaikka jQuery helpottaakin koodaamista huomattavasti. Omalla kohdallani toimintojen ymmärtäminen oli aiemmin paikoitellen hankalaa, koska en tun-

tenut JavaScriptiä hyvin. Koodiesimerkissä 34 verrataan JavaScriptin ja jQueryn eroa HTML-elementin valitsemisessa.

JavaScript:

```
var header = document.getElementById('header');
```

jQuery:

```
var header = $('#header');
```

KOODIESIMERKKI 34. HTML-elementin valitseminen

Toinen olennainen seikka on jQueryn selaintuki. Sovelluskehys toimii kaikilla selaimilla, joille toimeksiantaja takaa täyden, varmistetun toimivuuden. JQueryn versio 1 toimii Internet Explorerin versioon 6 asti. Uudempi versio 2 toimii pelkästään versiosta 8 ylöspäin. Kehyksen ympärillä kehitetään useita rinnakkaisprojekteja, kuten käyttöliittymiin keskittyvää jQuery UI:ta sekä mobiilisivustoja ja -sovelluksia varten tarkoitettua jQuery Mobilea. JQuery ei ainoa laatuaan, mutta näkemykseni mukaan se on nykyään suosituin vaihtoehdoista.

MVC-sovelluskehykset

MVC-malli (Model-View-Controller) on myös termi perinteisestä ohjelmistokehityksestä. Se on yhdenlainen suunnittelumalli (esitelty tarkemmin seuraavassa luvussa), joka jaottelee sovelluksen kolmeen osaan: dataan (malli, model), datan esittämiseen käyttäjälle (näkymä, view) ja käyttöliittymässä tapahtuviin toimintoihin (ohjain, controller). Käyttäjän toiminnot (esim. sivulta toiselle siirtyminen) välitetään ohjaimelle, joka määrittelee, mitä seuraavaksi tulisi tapahtua. Ohjain pyytää usein dataa mallilta (esim. sivun sisältö) ja antaa sen näkymälle käyttäjälle esitettäväksi (Snook 2009).

Front-end-kehityksessä JavaScriptillä toteutetut MVC-kehykset tulevat tarpeen, kun sivustossa tai sovelluksessa halutaan päivittää pelkästään tiettyjä osia näkymästä uudelleen AJAX:n avulla, ilman erillistä täyttä sivunlatausta (esim. lomakkeen tarkistusvirheiden näyttäminen). Olen havainnut, että koodista tulee

todella helposti sekavaa ja vaikeasti ymmärrettävää, jos sen organisointia ei suunnittele tarkasti tai siinä ei noudata jotain ennalta määrättyä mallia. Pelkästään jQueryn tai muun vastaavan kirjaston käyttö ei riitä, koska tällöin koodin toiminta perustuu liikaa elementtien manipulointiin ja JavaScript-tapahtumiin (events). Lisäksi muiden sovelluskehysten tavoin ne vähentävät runsaasti tarvetta kirjoittaa koodia täysin alusta.

Kaikki JavaScriptin MVC-sovelluskehykset eivät noudata yllä kuvattua mallia täysin, vaan joissain on omat ratkaisunsa toimintojen jaotteluun. Tätä voidaan kutsua MV*-malliksi joka tarkoittaa, että mukana on usein malli ja näkymä, mutta ei välttämättä ohjainta sellaisenaan. Ohjaimen vastuut voivat olla mm. sisällytettynä näkymään (Osmani 2012b). Esimerkiksi toimeksiantajan käyttämä Knockout (http://knockoutjs.com) noudattaa MVVM-mallia (Model-View-View-Model). En käy tässä työssä tarkemmin läpi eri kehysten ja mallien eroavaisuuksia, koska tässäkin tapauksessa eri vaihtoehtoja on jopa liiaksi asti, ja kehykset toteuttavat samoja toimintoja omilla tavoillaan. Tärkeämpää on tunnistaa sopiva malli sivuston tai sovelluksen tarpeiden perusteella.

Lyhyesti kuvattuna Knockout perustuu tarkasteltaviin muuttujiin ja data-sidontoihin (data-binding). Data-sidonnat annetaan merkkauksessa elementeille data-muuttujina, joiden arvoiksi asetetaan view model:in tarkasteltavia muuttujia. Kun muuttujat saavat uusia arvoja esim. käyttäjän toimien perusteella, Knockout ilmoittaa muutoksesta data-sidonnoille ja näkymä päivittyy ilman sivunlatausta. Alla olevassa esimerkissä personViewModel:n name-arvo muuttuu, kun käyttäjä kirjoittaa uuden nimen tekstikenttään. Uusi nimi päivittyy vanhan tilalle p-elementtiin. Koodiesimerkissä 35 on esimerkki Knockout-sovelluskehyksen view model:sta ja sen muuttujien käyttämisestä merkkauksessa.

JavaScript:

```
function personViewModel () {
  this.name = ko.observable('Foo');
  this.age = ko.observable('26');
}

ko.applyBindings(new personViewModel());
```

HTML:

```
<p>Name: <span data-bind="text: name"></span></p>
<label for="new-name">New name</label>
<input type="text" data-bind="value: name">
```

KOODIESIMERKKI 35. Knockout-sovelluskehyksen käyttö

8.2 Koodaustyyli

JavaScript on oikeana ohjelmointikielenä HTML:ää ja CSS:ää monimutkaisempi, joten en ole laatinut oppaaseen käytäntöjä jokaisesta mahdollisesta tyyliseikasta, vaan olen viitannut Googlen laatimaan JavaScript-tyylioppaaseen. (http://googlestyleguide.googlecode.com/svn/trunk/javascriptguide.xml). Se kattaa mielestäni hyvin joka projektissa noudatettavat yleiset säännöt syntaksista ja koodaustyylistä.

Suunnittelumallit

Osmani (2012b) näkee koodin toistuvien teemojen tunnistamisen sekä optimoinnin olevan yksi tärkeimpiä seikkoja ylläpidettävyyden kannalta. Tähän voidaan käyttää apuna ohjelmistokehityksestä tuttua tekniikkaa, suunnittelumalleja (design pattern). Ne ovat uudelleenkäytettäviä ratkaisuja yleisimmin esiintyviin ohjelmistokehityksen ongelmiin. Tekniikka ei ole uusi, vaan sen periaatteet ovat olleet käytössä alan alkupäivistä lähtien. Mallien suurimmat hyödyt voidaan tiivistää muutamaan kohtaan. Ne ovat todistettuja ratkaisuja, joita monet kehittäjät ovat käyttäneet ja muokanneet paremmiksi omien kokemustensa ja näkemystensä pohjalta. Niitä voidaan yleisluontoisuutensa ja dokumentoinnin laadun vuoksi soveltaa erilaisiin tarpeisiin. Koodin rakenteesta huolehtimisen sijasta kehityksessä voidaan keskittyä ratkaisun yleiseen laatuun, joka ehkäisee suurempiin ongelmiin johtavia pieniä virheitä. Mallien ennalta määritelty rakenne tekee koodista yhtenäisempää, auttaa sen organisoinnissa ja voi joissain tapauksissa pienentää rivimäärää vähentämällä saman koodin toistamista (Osmani 2012b).

Tutustuin tietoisesti ensimmäistä kertaa suunnittelumalleihin oppaan tekemisen yhteydessä ja verkkokauppaprojektia toteuttaessa. Olin tietämättäni hyödyntänyt niitä jo aiemmin, käyttäessäni jQuery-kirjastoa, jossa toimintoja on yksinkertaistettu perus-JavaScriptiin verrattuna facade-mallilla. Kyseisen mallin ideana on tarjota helppokäyttöisiä rajapintoja monimutkaisempiin toimintoihin (Osmani 2012b), kuten alaluvussa 8.1. kuvattuun HTML-elementin valitsemiseen (`$('#header');`).

Tarve mallien käyttöön tuli halusta välttää JavaScript-koodauksen huonoja käytäntöjä, anti-malleja (anti-patterns). Koodin testattavuuteen vaikuttavia anti-malleja kuvataan alaluvussa 8.3. Verkkokauppaprojektissa käytetään module-mallia, jolla organisoidaan koodia nimiavaruuksien (kokonaisuuksia yhteen liittyvää koodia yksittäisen tunnisteen alla) avulla ja vältetään samalla globaalien muuttujien ja olioiden määrää. Jos koodia ei erikseen määrittele tiettyyn nimiavaruuteen, tulee siitä window-olion alla eli globaalissa nimiavaruudessa sijaitsevaa. Muut kehittäjät tai kolmannen osapuolen kirjastot saattavat käyttää samoja muuttujien ja funktioiden nimiä, joka rikkoo toimintoja, jos kaikki koodi on globaalia. Alla tarkastellaan kahta eri tapaa module-mallin toteutukseen.

Object literal -notaatio

JavaScriptissä oliot voidaan kuvata avain/arvo-pareina object literal -notaatiolla. Literaalit voivat sisältää kaikkia JavaScriptin tietotyyppejä. Koodiesimerkeissä 36 ja 37 kuvataan object literal -notaatiota ja sen hyödyntämistä nimiavaruuksien määrittämisessä.

```
var objectLiteral = {
  variableKey: variableValue,
  functionKey: function () {
  ..
  };
}
```

KOODIESIMERKKI 36. Olio kuvattuna object literal -notaatiolla.

```
// global.js
var application = {
  utils: {},
  otherNamespace: {},
  …
}

// utils.js
application.utils.foo = function () {
  return 'bar';
}

// main.js
application.utils.foo();
```

KOODIESIMERKKI 37. Nimiavaruuksien määrittäminen ja laajentaminen

Moduulit

Module-malli määriteltiin alun perin tavaksi toteuttaa luokkien metodien ja muuttujien yksityisyyttä ja julkisuutta. JavaScriptissä moduulit ovat luokkia imitoivia olioita, joihin voi sisällyttää yksityistä ja julkista koodia. Kieli ei sisällä erikseen pääsymääritteitä (access modifier, esim. public, private tai protected). Ne ovat yksi tapa suojata koodia globaalilta nimiavaruudelta ja kirjoittaa testattavaa koodia (Osmani 2012b). Moduulista voidaan palauttaa literaali, joka sisältää kaiken julkiseksi tarkoitetun koodin. Koodiesimerkissä kuvataan moduulin määrittämistä sekä sen sisältämän koodin näkyvyyttä.

```
var module = function () {
    var privateVariable = 'foo',
      publicFunction = function () {
        return privateVariable;
      }

    return {
      public: publicFunction
    }
  },
```

```
  instance = new module();

var private = instance.privateVariable, // arvo on undefined
  public = instance.public();           // arvo on 'foo'
```

KOODIESIMERKKI 38. Moduulin määrittäminen ja käyttö

8.3 Laadun varmistus

Kuten luvussa 4 kerrottiin, kolmannen osapuolien kirjastoille kirjoitetut testit tuovat luotettavuutta ja varmuutta siitä, että kirjasto toimii myös tulevaisuudessa. Ne ovat automaattisesti ajettavia yksikkötestejä, joiden toteutukseen löytyy myös useita valmiita ratkaisuja. Yksikkötestauksella tarkoitetaan koodin funktioiden ja osa-alueiden, eli yksiköiden odotetun toiminnan testaamista. Testeillä määritellään, palauttaako funktio oikean arvon annetuilla parametreillä ja hoidetaanko mahdolliset virheet oikein, jos parametrit ovat virheellisiä. Ne auttavat tunnistamaan virheitä koodin toiminnassa ja parantavat sitä kautta koodin laatua. Jotta yksikkötestaus olisi mahdollista, koodin testattavuus on otettava huomioon ennen sen kirjoittamista. Funktioiden on oltava pienempiä sekä yksinkertaisempia ja pelkästään yhden toiminnon suorittavia, kuin laajoja ja useita toimintoja kattavia (McFarlin 2012).

Yksi esimerkki testaussovelluskehyksistä on QUnit (http://qunitjs.com), joka kuuluu jQueryn tuoteperheeseen. Kehyksen käyttö vaatii yhden JavaScript-dokumentin liittämisen mihin tahansa HTML-dokumenttiin, jossa testien tulokset halutaan esittää. Lisäksi mukana on tulosten ulkoasua varten yksi tyylitiedosto. Näkemykseni mukaan yleisin tapa on se, että testeille ja niiden tuloksille luodaan erillinen dokumentti, jolloin niitä ei tarvitse poistaa sivuston tai sovelluksen tuotantoversiosta. Koodiesimerkki 39 sisältää esimerkit QUnitin kehyksenä toimivasta HTML-dokumentista, testattavasta JavaScript-koodia ja sen testeistä. Kuvassa 9 nähdään kyseisten testien tulokset QUnitin kuvaamana.

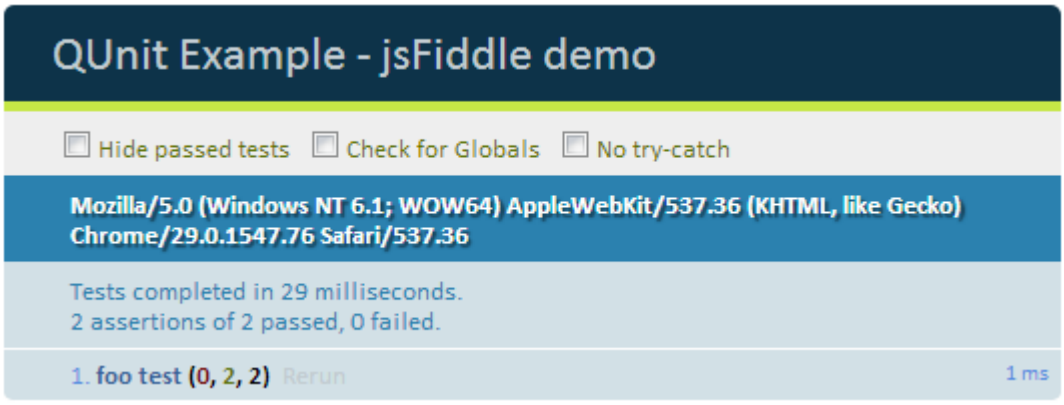
HTML:

```
<!DOCTYPE html>
<html>
<head>
  <link rel="stylesheet" href="/css/qunit.css">
  …
</head>
<body>
  <div id="qunit"></div>
  <script src="/js/vendor/qunit.js"></script>
  <script src="/js/main.js"></script>
  <script src="/js/tests.js"></script>
</body>
</html>
```

JavaScript:

```
// main.js
function foo (bar) {
  return bar;
}

// tests.js
test('foo test', function() {
  ok(foo('baz') === 'baz', 'Passed');
  ok(typeof foo('baz') === 'string', 'Passed');
});
```

KOODIESIMERKKI 39. QUnit-sovelluskehyksen käyttö



KUVA 9. QUnitin testitulos

Tomlinson (2013) kuvaa anti-malleja, jotka tekevät JavaScript-koodin testaamisesta vaikeaa ja jopa mahdotonta. Monitasoinen (esimerkiksi monia sisäkkäisiä toistolauseita), eri tasoilla useita toisten funktioiden kutsuja sisältävä koodi on vaikeaselkoista ja huonosti ylläpidettävää, eikä sitä voi testata yksikkönä. Alaluvussa 6.3. suositeltiin välttämään sisällytetyn JavaScriptin kirjoittamista, jota myös Tomlinson (2013) pitää anti-mallina. Parempi tapa on toimia yllä olevan esimerkin mukaisesti, ja linkittää testattavan koodin sisältävät dokumentit merkkauksessa. Vaikka koodi olisi omassa dokumentissaan, on sen oltava myös testien saavutettavissa. Esimerkiksi jQueryn `$(document).ready()`-funktion sisällä olevaa koodia ei voi testata koodiesimerkin 41 mukaisella tavalla. Tomlinson (2013) suosittelee module-suunnittelumallin käyttöä, jolla koodille voidaan määrittää julkisia rajapintoja testien käyttöön. Koodiesimerkissä 40 on JavaScript-funktio, joka ei ole testattavissa QUnitilla.

```
// main.js
$(function ($) {
  function foo (bar) {
    return bar;
  }
});

// tests.js
// Palauttaa Chromen kehitystyökaluissa virheen: Uncaught ReferenceError: foo
is not defined
test('foo test', function() {
  ok(foo('baz') === 'baz', 'Passed');
  ok(typeof foo('baz') === 'string', 'Passed');
});
```

KOODIESIMERKKI 40. QUnitin saavuttamattomissa olevaa koodia

Yksikkötestauksen lisäksi koodin laatua voidaan valvoa lint-sovelluksilla (esim. JSHint, http://www.jshint.com), jotka tarkastelevat syntaksia ja hyvien koodauskäytäntöjen noudattamista. Mielestäni käytännöllisin tapa on hankkia kehitysympäristöön tai tekstieditoriin lint-liitännäinen, joka ilmoittaa virheistä sitä mukaa kun koodia kirjoitetaan.

9 POHDINTA

Asetin ennen työn aloittamista henkilökohtaiseksi tavoitteekseni omien tietojeni ja taitojeni kasvattamisen. Se täyttyi mielestäni todella hyvin. Löysin monia tutustumisen arvoisia tekniikoita, jotka olivat itselleni aiemmin tuntemattomia. Toteutin työn ohessa verkkokauppasovellusta, jossa vastuualueekseni jäi enimmäkseen JavaScript-koodaaminen. Sain sen vuoksi erityisesti uutta osaamista ja näkemystä kyseiseen front-end-kehityksen osa-alueeseen. Osa työssä käsitellyistä tekniikoista jäi vaille kokeilua, koska niitä ei vielä jalkautettu projektissa. Kyseinen projekti kuitenkin jatkuu edelleen ja uusia tulee jatkossa, joten en usko niiden unohtuvan täysin. Työn kirjoittaminen olisi voinut sujua nopeammin, mutta hyvänä puolena pidän sitä, että ehdin saada paljon käytännön kokemusta ja näkemystä aiheesta. Toivon sen näkyvän työssä.

Esittelin oppaan nykyistä versiota kesällä, ennen vuosilomaani. Kollegat pitivät alkuesittelyssä kuvatuista, selkeistä tavoitteista, joihin käytännöillä pyritään, kolmannen osapuolen tekniikoiden esittelyistä ja koodiesimerkeistä. Esimerkkejä toivottiin jopa lisää. Toinen osio, JavaScript-testauksen käyttäminen, herätti myös mielenkiintoa ja testien kirjoittamista kaivattiin. Kommenttien perusteella voin päätellä, että oppaan ensimmäinen versio on onnistunut. Nähtäväksi jää vielä, kuinka paljon kehittäjät hyödyntävät yhteisiä käytäntöjä ja ohjeistuksia tulevissa projektissa. Ne ovat auttaneet itseäni jo paljon ja uskon, että verkkokauppasovelluksen front-end-koodista on tullut oppaan ansiosta laadukkaampaa.

Jatkokehitettävää löytyy silti vielä. Huomasin, että jatkossa vaaditaan enemmän tarkempaa, projektikohtaista ohjeistusta koodin kirjoittamisesta. Nykyinen oppaan taso ei vielä täysin riitä, vaan tällä hetkellä se on vielä pelkästään hyvä lähtökohta erilaisten sivustojen ja sovellusten front-end-kehitykselle. Jotta opasta olisi helpompaa päivittää, dokumentin sijasta parempi paikka sille olisi oma sivusto, jonka toteutukseen suunnittelin käyttäväni jotain HTML-muotoisen dokumentaation laatimiseen tarkoitettua työkalua. Toinen laajentamismahdollisuus on oppaan räätälöinti muiden tiimien käyttöön. Kyselin heidän tarpeistaan ja selvisi, ettei heillä joko ole vastaavia oppaita tai ne ovat vanhentuneita. Kesälomat estivät kuitenkin toistaiseksi yhteistyön.

LÄHTEET

Bewick, C. 2010. HTML5 Custom Data Attributes (data-*). Luettu 14.9.2013. http://html5doctor.com/html5-custom-data-attributes

CGI Group Inc. 2013a. CGI at a glance. Luettu 22.8.2013. http://www.cgi.com/sites/cgi.com/files/brochures/cgi_broc02_letter_cgi_at_a_glance.pdf

CGI Group Inc. 2013b. Image sprite. Asiakasprojektin materiaalia.

Essityöryhmä. 2003. Esteettömyyttä koskevat suositukset, standardit ja lait. Luettu 13.8.2013. http://appro.mit.jyu.fi/essikurssi/suositukset/t1

Gallagher, N. 2012a. About HTML semantics and front-end architecture. Luettu 27.8.2013. http://nicolasgallagher.com/about-html-semantics-front-end-architecture

Gallagher, N. 2012b. About normalize.css. Luettu 29.8.2013. http://nicolasgallagher.com/about-normalize-css

Gartner Inc. 2011. Gartner Reveals Top Predictions for IT Organizations and Users for 2012 and Beyond. Luettu 2.8.2013. http://www.gartner.com/newsroom/id/1862714

Gimmebar. 2013. Front end styleguides and pattern libraries. Luettu 15.8.2013. https://gimmebar.com/collection/4ecd439c2f0aaad734000022/front-end-styleguides-and-pattern-libraries

Google. 2012. Building Mobile-Optimized Websites. Luettu 23.8.2013. https://developers.google.com/webmasters/smartphone-sites

Holst, C. 2013. A Guide To Designing Touch Keyboards (With Cheat Sheet). Luettu 14.9.2013. http://uxdesign.smashingmagazine.com/2013/08/13/guide-to-designing-touch-keyboards-with-cheat-sheet

Howe, S. 2012. A Beginner’s Guide to HTML & CSS. Luettu 15.8.2013. http://learn.shayhowe.com/advanced-html-css/preprocessors

Keränen, V, Penttinen J. 2007. Verkko-oppimateriaalin tuottajan opas. Docendo. Jyväskylä.

Korpela, J, Lehdonvirta P. 2013. HTML5 sovellusalustana. RPS-yhtiöt. Helsinki.

Kuisma, K. 2012. Päätelaitteiden monimuotoisuus hallintaan Mobile first -strategialla. Luettu 22.8.2013. http://vierityspalkki.fi/2012/04/12/vieraskyna-paatelaitteiden-monimuotoisuus-hallintaan-mobile-first-strategialla

Long, J. 2012. I Don’t Speak Your Language: Frontend vs. Backend. Luettu 17.6.2013.
http://blog.teamtreehouse.com/i-dont-speak-your-language-frontend-vs-backend
Marcotte, E. 2010. Responsive Web Design. Luettu 27.10.2013.
http://alistapart.com/article/responsive-web-design

McFarlin, T. 2012. The Beginner’s Guide to Unit Testing: What Is Unit Testing? Luettu 28.9.2013.
http://wp.tutsplus.com/tutorials/creative-coding/the-beginners-guide-to-unit-testing-what-is-unit-testing

Meyer, E. 2011. CSS Tools: Reset CSS. Luettu 28.8.2013.
http://meyerweb.com/eric/tools/css/reset

Netflix Inc. 2012. Testing Netflix on Android. Luettu 2.8.2013.
http://techblog.netflix.com/2012/03/testing-netflix-on-android.html

Osmani, A. 2012a. Journey Through The JavaScript MVC Jungle. Luettu 25.9.2013.
http://coding.smashingmagazine.com/2012/07/27/journey-through-the-javascript-mvc-jungle

Osmani, A. 2012b. Learning JavaScript Design Patterns. Luettu 12.9.2013.
http://addyosmani.com/resources/essentialjsdesignpatterns/book

Otto, M. 2012. Building Twitter Bootstrap. Luettu 8.3.2013.
http://alistapart.com/article/building-twitter-bootstrap

Search Engine Land. 2010. It’s Official: Google Now Counts Site Speed As A Ranking Factor. Luettu 10.8.2013.
http://searchengineland.com/google-now-counts-site-speed-as-ranking-factor-39708

Smart Insights. 2013. Mobile Marketing Statistics 2013. Luettu 29.7.2013.
http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics

Snook, J. 2009. JavaScript MVC. Luettu 25.9.2013.
http://alistapart.com/article/javascript-mvc

StatCounter Global Stats. 2013. Top 12 Browser Versions (Partially Combined) in Finland on 2013. Luettu 17.9.2013.
http://gs.statcounter.com/?chart_type=line&statType_hidden=browser&region=Finland&region_hidden=FI#browser_version_partially_combined-FI-yearly-2013-2013-bar

SuomiSanakirja. 2013. Semantiikka. Luettu 15.8.2013.
http://www.suomisanakirja.fi/semantiikka

Swanson, L. 2013. Improving UX Through Front-End Performance. Luettu 10.8.2013.
http://alistapart.com/article/improving-ux-through-front-end-performance

Tomlinson, S. 2012. The Three Cs of Front End Performance – Concatenate, Compress and Cache. Luettu 11.8.2013.
https://shanetomlinson.com/2012/front-end-performance-concatenate-compress-cache

W3C. 2013a. Differences from HTML4. Luettu 27.8.2013.
http://www.w3.org/TR/html5-diff

W3C. 2013b. Elements: Embedding custom non-visible data with the data-* attributes. Luettu 14.9.2013.
http://www.w3.org/TR/2011/WD-html5-20110525/elements.html#embedding-custom-non-visible-data-with-the-data-attributes

W3C. 2013c. Elements: The class attribute. Luettu 14.9.2013.
http://www.w3.org/TR/2011/WD-html5-20110525/elements.html#classes

WebAIM. 2013. United States Laws: Overview of the Rehabilitation Act of 1973 (Sections 504 and 508). Luettu 13.8.2013.
http://webaim.org/articles/laws/usa/rehab

Web Platform Docs. 2013. Web accessibility basics. Luettu 13.8.2013.
http://docs.webplatform.org/wiki/concepts/accessibility

Web Site Optimization. 2008. The Psychology of Web Performance. Luettu 5.8.2013.
http://www.websiteoptimization.com/speed/tweak/psychology-web-performance

Yahoo! Inc. 2013. Graded Browser Support. Luettu 5.8.2013.
http://yuilibrary.com/yui/docs/tutorials/gbs

LIITTEET

Liite 1. Front-end Development Guide

Front-end Development Guide

Table of Contents

1 Overview

This document contains general guidelines and best practices for developing the front-end of eBusiness websites and -applications.
The main goals are to 1) achieve good, consistent quality in the markup, styling & JavaScript, 2) make the code maintainable, readable and faster to develop further and 3) help new developers in joining the project teams. The golden rule to follow is that the whole front-end code base should look like it's written by one person.

If you're joining a project, the existing guidelines and code style should be followed over the guide's.

This is a living document that is meant to evolve and change over time, as new standards are found and agreed upon. Feel free to comment on the contents and contribute with your own ideas.

Further explanation and reading about the practices selected to this guide is provided by links to other resources.

2 General guidelines

2.1 Third-party resources

Using third-party resources is encouraged to shorten development and testing times. When selecting resources, different choices should be evaluated and tested before choosing the one to use. Resources that aren't actively developed or are poorly documented shouldn't generally be used. One good indicator whether to use one resource over others is its GitHub activity.

Use full libraries and plugins in development so debugging doesn't become impossible. Compression and concatenation is done in the build process.

Listed below are some of the most important resources used in projects and reasons why we've opted to use them.

Twitter Bootstrap

An extensive front-end framework containing several common UI components and base styles. It provides a solid, high quality codebase and many good practices to use in our own front-end development. We use the 3.x WIP version instead of the 2.x master. The documentation isn't hosted online at the moment, and must be run locally on a Jekyll server.

https://github.com/twitter/bootstrap/tree/3.0.0-wip

LESS

Our choice of CSS preprocessor, Twitter Bootstrap uses it. No sense in writing regular CSS when there dynamic features such as variables, mixins and nesting available.

http://lesscss.org

Jade Template language

Markup is written in Jade language in Java projects, instead of JSP. Jade's syntax is much more reader-friendly and faster to write. Templates are separated by functionalities.

http://jade-lang.com

HTML5 Boilerplate

The base Jade template is adapted from HTML5 Boilerplate, because it utilizes several good, tested principles and techniques.

http://html5boilerplate.com

jQuery

Version 2.0 is used in projects where older browsers (read: Internet Explorer) aren't a concern. Otherwise use earlier versions.

Knockout.js (or any other MVC library/framework)

Creating AJAX-heavy UIs can quickly make the JS code very complex and unwieldy. It's better to use client-side MVC in these cases. The project's requirements should dictate the used solution. Knockout is a good starting point if you're unfamiliar the technique.

http://knockoutjs.com/

Reasons to use client-side MVC and tips on choosing the right one:
http://blog.stevensanderson.com/2012/08/01/rich-javascript-applications-the-seven-frameworks-throne-of-js-2012/

2.2 Responsive design and mobile-first

We utilize responsive design with mobile-first approach to ensure that the same UI is usable and accessible in as many end user devices as possible. Mobile-only interfaces are only made when the project calls for one.

Primers on responsive design and mobile-first:

http://johnpolacek.github.io/scrolldeck.js/decks/responsive/
http://bradfrostweb.com/blog/mobile/the-many-faces-of-mobile-first/

2.3 Document structure and naming

- Place all front-end related document under ../static and in their respective folders.
- Name documents by their main function and in English.
- Use dashes instead of camelCasing/PascalCasing or underscores.

Example:

2.4 Format

- Indent with two soft spaces.
- Single quotes in Jade and JS, double in LESS.
- Improve readability by inserting empty lines after block-level elements in markup, CSS selectors and code blocks in JS.
- Keep the line length reasonably short, around 80-120 columns.
- Avoid abbreviations and use intuitive and descriptive names in code.

Example:

2.5 Commenting

Ensure your code is descriptive, well commented and approachable by others. Comments should convey context or purpose and should not just reiterate a component or class name.

Bad example:

```
/* Modal header */
.modal-header…
```

Good example:

```
/* Wrapping element for .modal-title and .modal-close */
.modal-header…
```

3 Markup

Base the markup on a quality template such as HTML5 Boilerplate, which utilizes a number of best practices to aid in cross-browser functionality and mobile-first coding.

3.1 Doctype

We use the shortened version-less doctype because HTML5 markup is used.

```
!!!
doctype !!!
!doctype html
```

3.2 Format

- Keep the Jade templates clean and light.
- Separate larger sections such as headers and footers to their own documents and use them as blocks.
- Use mixins for common functionalities used in multiple templates or for easier maintenance as they are contained in a single file.
- Structure the content in a logical way and use headings hierarchically.

Example

base.jade:

```
include ../mixins/common-mixins
!doctype
html(lang='en')
  head
    block title
      | ESSI
  …

  body
    header
```

```
    include header
    include main-navigation


  .content
    block content

  footer
    include footer
```

header.jade:

```
header
 a.logo(src='logo.jpg', alt='ESSI')
 …
```

index.jade:

```
extends base

block prepend title
 Home

block content
 h1 Home Page!
 …

 h2 Article!
 article
 +commentForm(…)
 …
```

common-mixins.jade:

```
mixin commentForm(…)
 form.form-horizontal
```

…

Use elements in their semantic meaning:

- lists for presenting lists of elements
- tables for tabular data, not layout
- paragraphs instead of br-elements
- etc.

Utilize the new HTML5 elements and attributes when appropriate, but keep in mind that the spec is still under construction so some elements maybe removed or changed. The new input-types should also be used for better user experience in mobile devices. Use polyfills and fallbacks to provide the same functions across different browsers.

http://uxdesign.smashingmagazine.com/2013/08/13/guide-to-designing-touch-keyboards-with-cheat-sheet/

Use alt- or desc-attributes in images (empty if the image is used solely for layout), captions & summaries for table, and titles for image-only links. Define tabindex for clickable non-button/link elements. Every element with an event should be focusable with a keyboard.

Attribute order:

1. class
2. id
3. data-*
4. other

h1.class#uniqueDiv(data-attr=’value’, attr=’value’)

3.3 WAI-ARIA

WAI-ARIA is a new specification regarding the accessibility of JavaScript rich sites and web applications. Best practices of its use are still being evaluated.

https://developer.mozilla.org/en-US/docs/Accessibility/ARIA

3.4 Microdata and data-attributes

Microdata is a set of attributes used by search engines to understand the content of webpages better. Schema.org provides vocabularies for different content types. How and where to use Microdata is still being evaluated.

http://schema.org

Data-attributes are used to serve metadata to JavaScript. Use them for data that isn't meaningful to a user, or when there are no other appropriate elements or attributes that can be used instead. Namespace the attributes to avoid conflicts with third-party JavaScript:

data-namespace-attr

Article about data-attributes and their usage:
http://html5doctor.com/html5-custom-data-attributes

3.5 Inline styling and JavaScript

Don't use inline styles or JS in script tags or attributes, except for client-side templating. Use classes, IDs and data-attributes as hooks to JS instead.

Bad example:
a(href='#', onclick='clickLink()', style='font-size: 13px') Link!

```
script
  var inlineVar = 1;
  function clickLink() {
    …
  }
```

4 Styling

https://github.com/csswizardry/CSS-Guidelines

4.1 Box model

Box model denotes the HTML element's dimensions - width, height, border, padding and margins.

To enforce consistent cross-browser rendering, set box-sizing-rule for all elements.

```
* {
  box-sizing: border-box;
}
```

4.2 Format

- Add an empty line between rules.
- Start every declaration on its own line within the selector.
- Keep each selector is on its own line in multi-selector rules.
- Use lowercase and shorthand hex values, e.g. #fff instead of #FFFFFF
- Use double quotes.
- Quote attribute values in selectors, e.g. input[type=”text”].
- Avoid units for zero-values when allowed, e.g. margin: 0; instead of margin: 0px;
- Break up particularly long values to multiple lines.
- Group properties together with shorthand rules.
- Don't abuse !important rule to force specificity.
- Keep selectors short and limit the number of elements.
- Use px over em for font size.
- Keep line-height unit-less.
- Solve compatibility problems with valid rules, not hacks

Example:

```
.class,
.other-class,
.class[type=”text”] {
  margin: 0 10px;
  font-size: 13px;
  color: #aaa;
}
```

4.3 Naming and selectors

- Adapt Bootstrap’s and the OOCSS naming and selector practices to suit the project
- Use informative, non-abbreviated names that are structural rather than presentational.
- Use classes and IDs over generic element tags. IDs should be used for elements that occur only once in a page.
- Replace spaces with dashes in classes, use camelCasing in IDs.
- Prefix classes based on the closest parent component's base class.

Example:

```
.product
.product-list
.product-price
.btn-save
.btn-cancel
#modalLogin
```

OOCSS: http://coding.smashingmagazine.com/2011/12/12/an-introduction-to-object-oriented-css-oocss/

On specificity: http://www.slideshare.net/stubbornella/our-best-practices-are-killing-us

4.4 Ordering declarations

Order declarations by their purpose:

```
.class {
  // Extended classes and mixins
  .extended-class;
  .mixin();

  // Positioning
  position: absolute;
  top: 0;
  z-index: 100;

  // Display & Box Model
  display: block;
  float: left;
  width: 100px;

  // Typography
  font-size: 14px;
  line-height: 1.5;

  // Visual
  color: #222;
  border: 1px solid;

  // Other
  opacity: 1;

  // Nested rules
  .nested-class …
}
```

4.5 Using LESS

- Utilize Bootstrap's ready-made variables, mixins and helper classes.
- Create custom variables, mixins and helper classes to keep the code DRY and easier to maintain.
- When using nesting, don't nest more than two-three levels deep. If you need more levels, refactoring should be considered.
- Avoid a large number of nested rules (more than 20 lines).
- If you define common custom components used throughout the markup, consider avoiding the multiple class -pattern by LESS extending:

button.btn.btn-primary.btn-large.btn-save

→

```
.btn-save {
  .btn;
  .btn-primary;
  .btn-large;
}
```

button.btn-save

Read the SASS style guide for more good principles:

http://css-tricks.com/sass-style-guide

4.6 Organizing documents

- Define imports at the beginning of the document.
- Custom variables, mixins and utility classes should be next or in separate documents.
- Organize sections by component.
- Mark the sections with comment blocks and break down large sections.

Example:

```
@import tablets.less

// VARIABLES
// ---------
@color: #B45455

// HEADER
// ------
…

// COMPONENTS
// ----------
// Buttons
…
// Tables
…
```

5 JavaScript

5.1 Code style

Instead of copying one of the many great code style guides here, we've elected to follow Google's style guide:

http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml

- Implement back/forward and deep linking support in sites where content is loaded with AJAX with libraries such as Sammy.js or History.js

http://sammyjs.org/
https://github.com/browserstate/history.js

- Use namespacing to reduce the amount of globals and to prevent conficts with third-party resources.
- Utilize the Modernizr library when detecting browser feature support, rather than relying on the user-agent string.
- Don't mix markup and JS code. Use client-side templating or create DOM objects with jQuery.

Bad example:

```
var div = '<div class="value" attr="value">text</div>';
```

Good example:

```
var div = $('<div />', {
  'class': 'value',
  'attr': 'value',
  'text': 'text'
});
```

- Construct object literals when several function parameters are needed.

Bad example:

```
function submit(param1, param2, a, b, c, d);
```

Good example:

```
var params = {
  param1: 1,
  param2: 2
  …
}

function submit(params);
```

- Use a single var pattern.

Bad example:

```
var a;
var b;
var c;
```

Good example:

```
var a,
  b,
  c;
```

5.2 Code quality

Write testable, scalable and well-maintainable JavaScript. The same standards of quality should apply to both front- and back end-code. Use a JSLint plugin for your preferred IDE or text editor to detect syntax errors.
If the project requires a large amount of JS functionality, use design patterns and avoid the anti-patterns outlined in the following articles:

http://alistapart.com/article/writing-testable-javascript

https://shanetomlinson.com/2013/testing-javascript-frontend-part-1-anti-patterns-and-fixes/ & Part 2

Tools and techniques for testing:
http://www.slideshare.net/emwendelin/javascript-ci

6 Accessibility

Achieving accessibility isn't an easy or fast task. Following this guide's practices is merely a good start. The UIs should meet the WCAG standards, where it makes sense in context of the project.

http://webaim.org/standards/wcag/checklist

7 Performance

- Set up minification, concatenation and compression to the build process to reduce HTTP requests and resource size.
- Follow Google's other best practices on performance optimization:
  https://developers.google.com/speed/docs/best-practices/rules_intro
  https://developers.google.com/speed/articles/
- Use Google's PageSpeed Insights, Chrome Dev Tools' Audits or YSlow plugin for Dev Tools and Firebug to meter performance and identify flaws.
- Utilize icon sprite sheets and icon fonts
- Optimize images for web consumption

8 Browser support

A consistent, quality user experience must be achieved on most common desktop and mobile browsers. Sites don't need to render the same across all browsers, so there's no sense in trying to go for pixel-perfection. A better alternative is to consider progressive enhancement by offering a solid baseline to users with older browsers while providing a richer UI to those with modern browser.

http://alistapart.com/article/understandingprogressiveenhancement

Baseline for desktop/mobile browser testing:

- Internet Explorer 8.0 - 10.0 (other versions by project-basis)
- Chrome: latest
- Firefox: latest
- Safari: iOS 5 & 6, latest desktop
- WebKit: Android 2.3 & 4

9 Further reading

The standards and best practices described in this document are based on similar guides:

Bootstrap Code Guide
https://github.com/mdo/code-guide

Isobar Front-end Code Standards & Best Practices
http://isobar-idev.github.com/code-standards

Front End Dev Guidelines
http://taitems.github.com/Front-End-Development-Guidelines

Idiomatic HTML, CSS and JS
https://github.com/necolas/idiomatic-html
https://github.com/necolas/idiomatic-css
https://github.com/rwldrn/idiomatic.js

10 Resources

https://github.com/twitter/bootstrap/tree/3.0.0-wip
http://lesscss.org
http://jade-lang.com
http://html5boilerplate.com
http://knockoutjs.com/
http://blog.stevensanderson.com/2012/08/01/rich-javascript-applications-the-seven-frameworks-throne-of-js-2012/
http://johnpolacek.github.io/scrolldeck.js/decks/responsive
http://bradfrostweb.com/blog/mobile/the-many-faces-of-mobile-first
http://schema.org
http://html5doctor.com/html5-custom-data-attributes
http://css-tricks.com/sass-style-guide
http://coding.smashingmagazine.com/2011/12/12/an-introduction-to-object-oriented-css-oocss/
http://www.slideshare.net/stubbornella/our-best-practices-are-killing-us

http://google-styleguide.googlecode.com/svn/trunk/javascriptguide.xml
http://sammyjs.org/
https://github.com/browserstate/history.js
http://alistapart.com/article/writing-testable-javascript
https://shanetomlinson.com/2013/testing-javascript-frontend-part-1-anti-patterns-and-fixes
http://webaim.org/standards/wcag/checklist
http://alistapart.com/article/understandingprogressiveenhancement